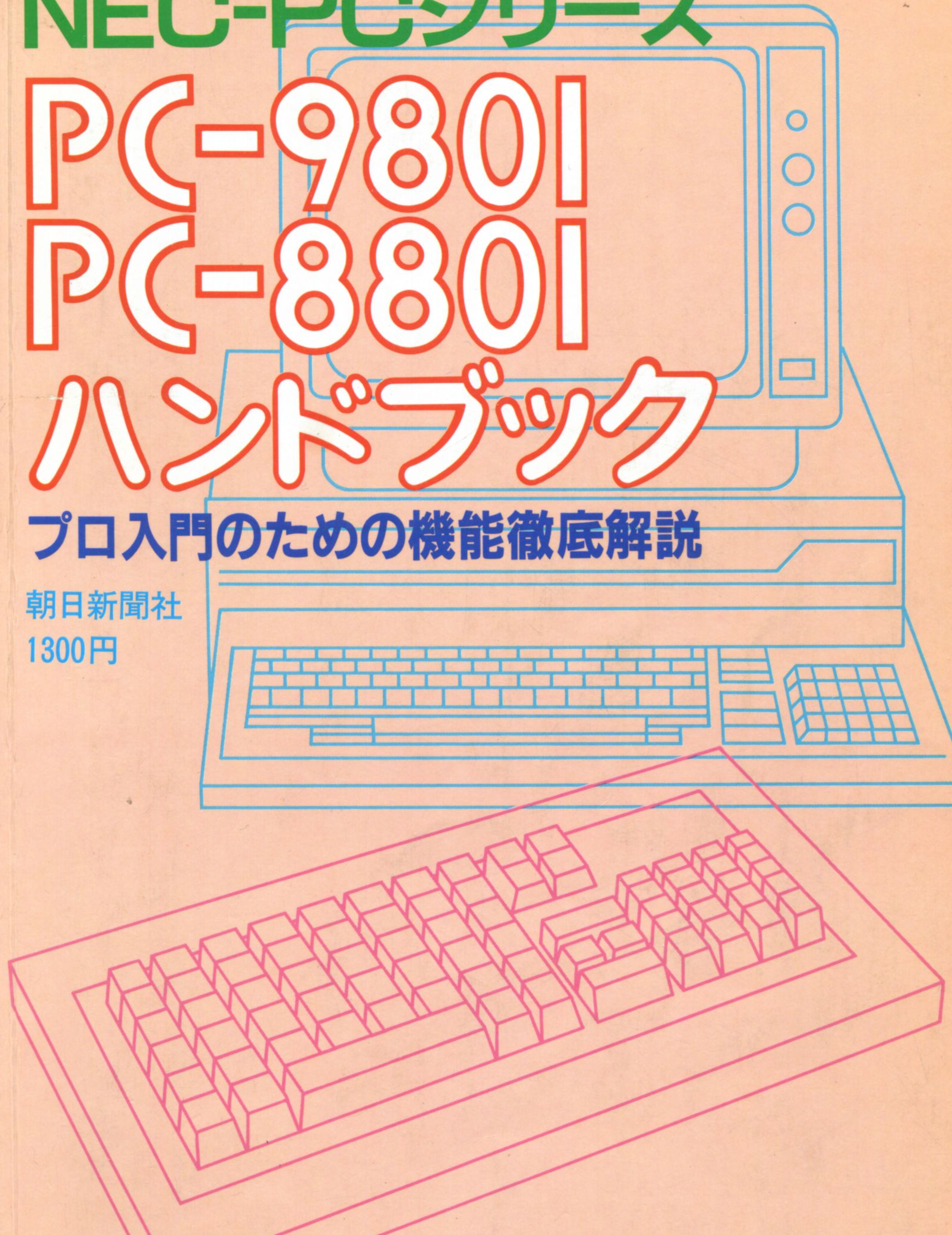
NEC-PCシリーズ

PC-9801 PC-8801 ハンドブック

プロ入門のための機能徹底解説



朝日新聞社

1300円

朝日新聞電子計算室編

まえがき

本書は、ＮＥＣのパソコン「ＰＣ－９８００」および「ＰＣ－８８００」の持つ機能を、最大限に引き出すためのハンドブックです。２冊にすべきものを１冊にまとめ、比較できるようにしてあります。

現在、市場にはそれぞれ特徴を持ったパソコンがあふれています。利用者はそのなかから、各自のシステムに合ったパソコンを探し出せばよいわけです。お断りしておきますが、上記両機種が「最良の機種だから採り上げた」という意味はありません。たまたま、われわれが構築を予定したシステムになじむものだっただけであり、通信機能などはＮＥＣ側としても、これから相当な強化をはかるでしょう。

われわれは、マニュアル類の良否を機種選択の大きなポイントにしていますが、まだまだ、そこに気づかない開発担当者が多いようです。分冊になったマニュアルで時間を浪費するくらいなら、最初に「自分にあったハンドブック」を作ってから仕事を始めた方が能率的で、より現実的です。

以上が本書を作った理由です。目次を見れば分かるとおり、本書は次の諸点に留意しました。

①　全体の構成から準備作業までをひとまとめにして、いつでも参照できるようにした。

②　特に、機械語の解説に力を入れた。

③　ＢＡＳＩＣの例題プログラムは、事務処理に必要な基本的アルゴリズムをすべて網羅した。

④　機械語のプログラムは、初心者のためを思い、一行ごとにコメントをつけた。

⑤　９８００用表示選択方式による日本語入力をとりあげた。

表現上の問題として、アセンブラー・機械語関連の「ニーモニック表記」はすべてインテル社のニーモニック表現にしました。われわれは「ニーモニック（表記記号）に著作権は成立しない」と考えます。むろん特定のニーモニックで記述したプログラムには、著作権はあるでしょう。しかし、ニーモニックそのものは人類共通の財産であり、また、そうしないと進歩も普及もあり得ません。われわれはインテル社とザイログ社のニーモニックで作業を進め、その対照表まで作りましたが、最終的にはインテル社のものに統一しました。むろんインテル社の了解は得ています。またザイログ社のニーモニックについては、今回、本書では触れませんでした。ザイログ社の話では、これは当然ですが、普及のためのニーモニックの機能解説にまで制限を加える積もりはないとのことです。

ＮＥＣから、多大のご支援をいただきました。心から感謝して、お礼を申しあげる次第です。

最後に、本書はあくまで本書に記述してある範囲でご使用いただければ幸いです。極めて突っ込んだ技術解説がしてありますので、ひとつひとつ読者のご質問にお答えするのは物理的に不可能です。まことに勝手ですが、ご質問はご遠慮していただきたく、あらかじめお願い申し上げます。

朝日新聞社電子計算室

目　次

Ⅱ・ＰＣ－９８０１ハード・ソフト解析編

Ⅲ・ＰＣ－８８０１ハード・ソフト解析編

Ⅳ・共通編

本書の使用法

（1）本書は次の４編から成る。

「Ｉ・基礎編」　主として両機種共通の機能や使用法

「Ⅱ・ＰＣ－９８０１ハード・ソフト解析編」

「Ⅲ・ＰＣ－８８０１ハード・ソフト解析編」

「Ⅳ・共通編」　エラーメッセージ・キャラクターコード表・日本語ＪＩＳコード表

（2）本書で扱うＢＡＳＩＣは、すべてＮ８８－ＢＡＳＩＣである。
Ｎ－ＢＡＳＩＣを動かすには、ＰＣ－９８０１ではＮ－ＢＡＳＩＣ用のシステムディスクを使う。
ＰＣ－８８０１では、「ＮＥＷ　ＯＮ」コマンドでＮ－ＢＡＳＩＣに切り替える。

Ｎ－ＢＡＳＩＣの使い方については、姉妹書「パソコンＰＣシリーズ　８００１　６００１ハンドブック」を参照していただきたい。

（3）本文中に（９８）の表示がある場合は、ＰＣ－９８０１だけの機能や使用法を示す。
（８８）の表示がある場合は、ＰＣ－８８０１だけの機能や使用法を示す。
なにも表示がなければ、ＰＣ－９８０１とＰＣ－８８０１の両機種の機能や使用法を示す。

（4）（ディスク）の表示がある場合は、Ｎ８８　ＤＩＳＫ－ＢＡＳＩＣの機能や使用法を示す。

（5）ＢＡＳＩＣのコマンドやステートメントの書き方の説明は、次の規則にしたがう。

①〔　〕で囲んだ部分は指定しても指定しなくてもよい。

②＜　＞で囲んだ部分は指定する。

③〔，＜前の項目＞〕〔，＜あとの項目＞〕で、前の項目を指定せずにあとの項目を指定する場合、前の〔　〕の中のカンマ（，）も指定する。

（例）　ＣＯＬＯＲ〔＜ファンクションコード＞〕〔，＜バックグラウンドカラー＞〕〔，＜ボーダーカラー＞〕〔，＜フォアグラウンドカラー＞〕
でバックグラウンドカラーを０、フォアグラウンドカラーを７と指定する場合、次のようにする。
ＣＯＬＯＲ　，０，，７

④セミコロン（；）、ハイフン（―）、等号（＝）、引用符（”）などの記号は、指示された位置に記入する。

⑤反復記号（・・・・）の前の＜　＞、〔　〕で囲んだ部分は１行の長さ（２５５文字）以内でくりかえし指定できる。

（6）文章中にすみつきパーレン【　】で囲んだキーは、実際にキーを押す操作を表す。
また、【前のキー】＋【あとのキー】とある箇所は、前のキーを押しながら、あとのキーを押す操作を表す。【ＲＥＴ】はリターンキーを押す操作を表す。

（例）ＰＲＩＮＴ　３．１４１５９＊１０【ＲＥＴ】
↑――リターンキーを押す

（例）【ＣＴＲＬ】＋【ＸＦＥＲ】
――ＣＴＲＬキーを押しながらＸＦＥＲキーを押す

（7）１６進数を表すのに、ＢＡＳＩＣについての説明では、数字の頭に＆Ｈをつける。
文章中で番地などを表すときに使う１６進数には、数字の末尾にＨをつけて１０進数と区別する。

（8）２進数の１桁（ビット）の説明では、次のように頭に小文字のｂをつけて表す。

ｂ７	ｂ６	ｂ５	ｂ４	ｂ３	ｂ２	ｂ１	ｂ０

（9）「Ⅰ・11　ＰＣ－９８０１　表示選択方式による日本語入力」をする場合は、Ｎ８８－日本語ＢＡＳＩＣのシステムディスクが必要。

（１０）索引はコマンドやステートメントだけでなく、「機能」からも引けるようにした。

（参考）

２進数と１０進数と１６進数の関係は次のとおり。
２進数の４桁が１６進数の１桁に当たる。
１０進数の10、11、12、13、14、15は、16進数では英大文字のＡ、Ｂ、Ｃ、Ｄ、Ｅ、Ｆで表す。

１０進数	２進数	１６進数
０	０００００００００	０
１	０００００００１	１
２	０００００００１０	２
３	０００００００１１	３
４	００００００１００	４
５	００００００１０１	５
６	００００００１１０	６
７	００００００１１１	７
８	０００００１０００	８
９	０００００１００１	９
１０	０００００１０１０	Ａ
１１	０００００１０１１	Ｂ
１２	０００００１１００	Ｃ
１３	０００００１１０１	Ｄ
１４	０００００１１１０	Ｅ
１５	０００００１１１１	Ｆ

１２８の位 …… １の位
６４の位 …… ２の位
３２の位 …… ４の位
１６の位 …… ８の位

２進数の１１０１１００１は、１０進数では２１７、１６進数ではＤ９になる。

（１０進数）

１１０１１００１

１の位　１×　１＝　１
８の位　１×　８＝　８
１６の位　１×１６＝　１６
６４の位　１×６４＝　６４
１２８の位　１×１２８＝　１２８

２１７

（１６進数）

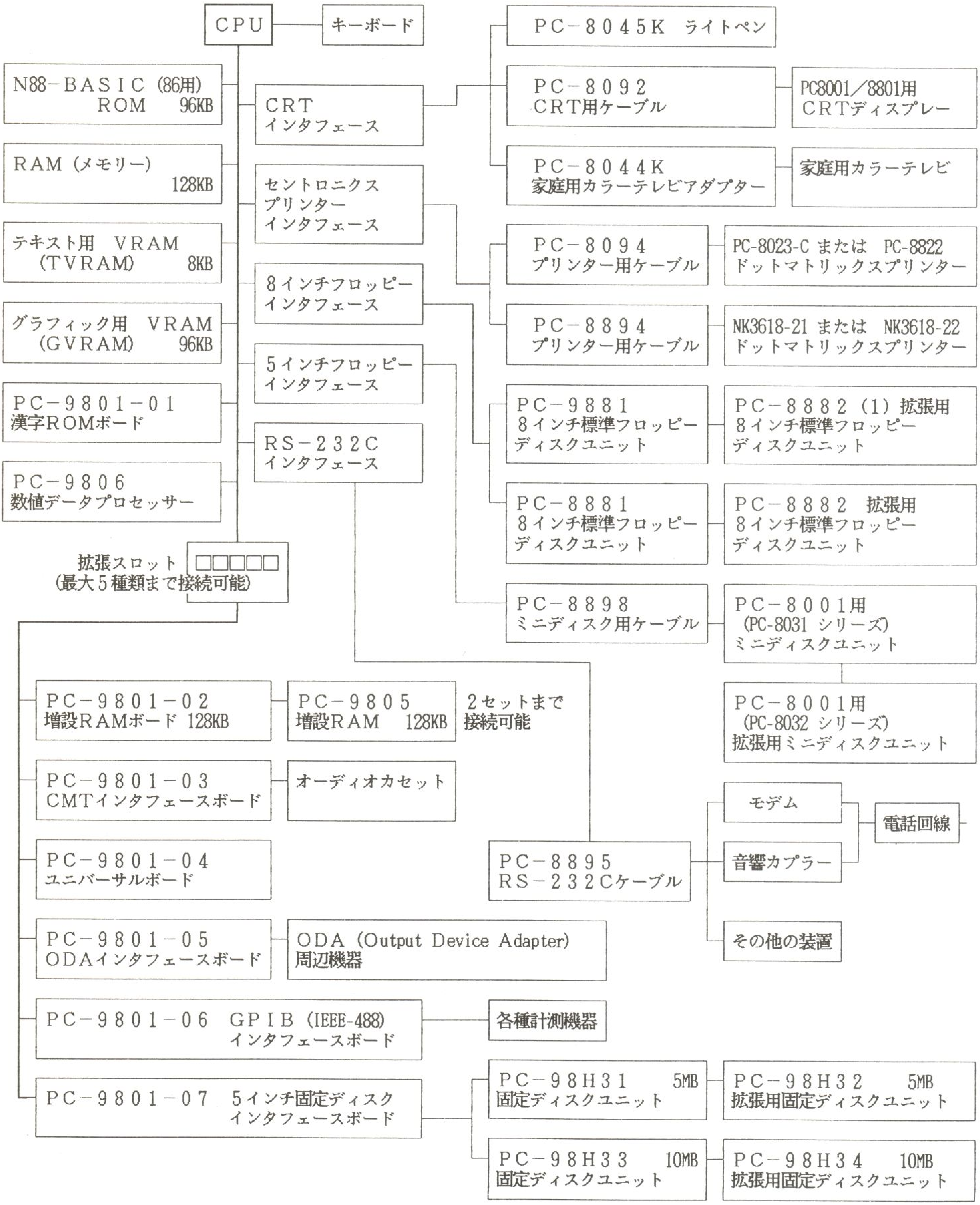
１１０１１００１

Ｄ　９

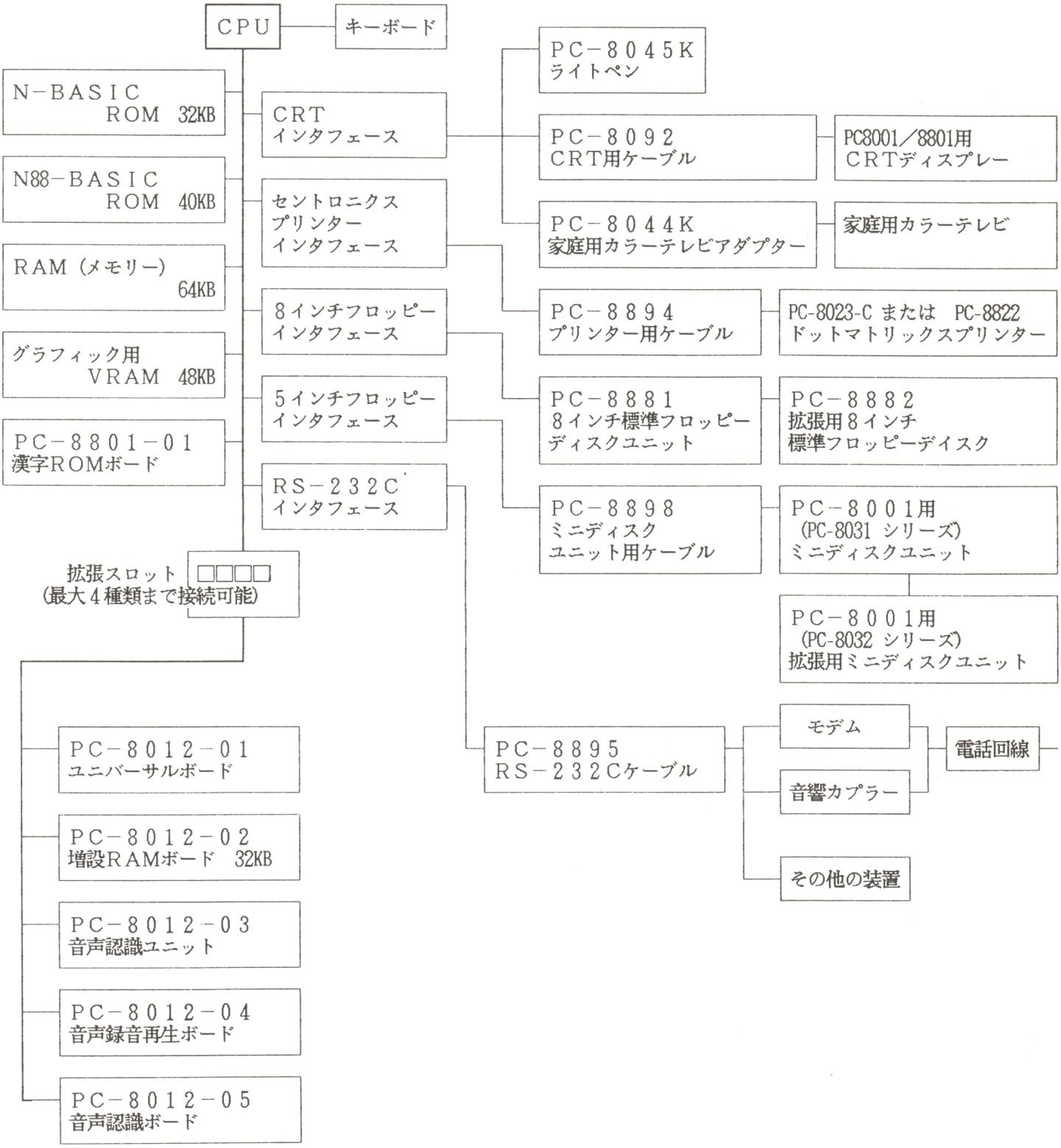
１の位　９×　１＝　９
１６の位　１３×１６＝　２０８

２１７

ハードウエア構成図　PC-9800シリーズ



ハードウエア構成図　PC-8800シリーズ



準備作業

☆装置の接続☆

ＰＣ本体後面にあるコネクタの表示を確認して、それぞれ対応するケーブルを接続する。ケーブル、コネクターに▼印がある場合には印を合わせる。キーボードは、ＰＣ本体前面、電源スイッチの下に接続する。拡張スロットには、増設ＲＡＭボードや漢字ＲＯＭなど必要に応じてセットする。故障の原因になるので、ケーブルの接続やディップスイッチなどを切り替えるときは、かならず各装置の電源スイッチを切ること。

ソケットの表示	接続する装置
Ｂ／Ｗ	モノクロディスプレー
ＣＯＬＯＲ	カラーディスプレー
ＲＳ－２３２Ｃ	RS-232C コネクター（通信用）
ＰＲＩＮＴＥＲ	プリンター
Ｇ	グラウンド端子
(98) ８”FLOPPY DISK	８ｲﾝﾁフロッピーディスク
(98) ５”FLOPPY DISK	ミニフロッピーディスク
(98) ８”FLOPPY DISK DC OUT	８ｲﾝﾁフロッピーディスク用電源
(98) AC-100V-3A	ディスプレー、ミニフロッピーディスク等用電源
(98) AC-100V	ＰＣ－９８０１本体用電源コード
(88) AC-100V	ＰＣ－８８０１本体用電源コード
(88) FLOPPY DISK	ミニフロッピーディスク
(88) ＣＭＴ	オーディオカセットレコーダ

☆ディップスイッチなどのセット☆

装置の設置、ケーブルの接続が終了したら、ディップスイッチ等をセットする。ここでは、ＢＡＳＩＣモードで標準的な使い方をする場合のセット方法を説明する。

《ＰＣ－９８０１》
①ディップスイッチ
システムの構成に合わせて、本体後面のディップスイッチとメモリースイッチをセットする。ディップスイッチは、下がＯＮ。ＳＷ１の１は専用高解度ディスプレーを使うときはＯＮ、標準ディスプレーではＯＦＦにする。ＳＷ１の２～６は白黒ディスプレーの画面を決めるスイッチで、２～５をＯＦＦ、６をＯＮにする。ＳＷ１の７～１０はＲＳ－２３２Ｃ（通信機能）の制御に使うが、７と９をＯＦＦ、８と１０をＯＮにしておく。ＳＷ２の１と２はＯＦＦ、３と４はＯＮ（８０字／行、２５行／画面）５はＯＮ、６～８はＯＦＦにする。ここまでセットしたらＰＣ－９８０１以外の周辺装置から順次電源を入れ、最後にＰＣ－９８０１の電源を入れる。Ｈｏｗ　ｍａｎｙ　ｆｉｌｅｓ（０－１５）？という表示（注）が出たらＲＥＴキーを押す。表示が出ないときは、電源を切り、装置の接続とディップスイッチをチェックする。

？	上	上	上	上	下	上	下	上	下
1	2	3	4	5	6	7	8	9	10

ＳＷ１

？は上下どちらでも可

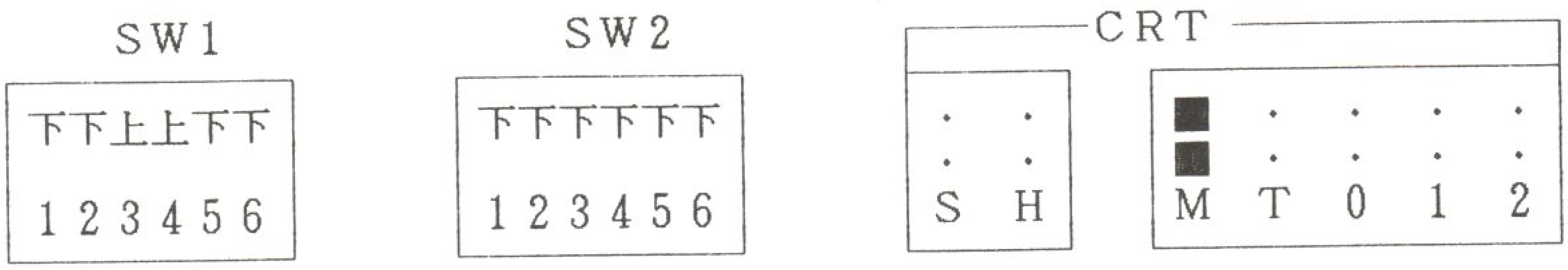
上	上	下	下	下	上	上	上
1	2	3	4	5	6	7	8

ＳＷ２

②メモリースイッチ
次に、不揮発性ＲＡＭを使ったメモリースイッチをセットして、システムの使用状態を決定する。ここでは、実装ＲＡＭの大きさを登録する方法を説明する。標準実装ＲＡＭ（１２８ＫＢ）のときはセット不要。まず、ＭＯＮと入力し、ＲＥＴキーを押すとモニターモードになる。次にＳＳＷ３と入力し、ＲＥＴキーを押すと００―と表示されるので、対応する数値を入力してＲＥＴキーを押す。これでセットは終わる。ＣＴＲＬキーを押しながらＢを押すとＢＡＳＩＣモードにもどる。

実装ＲＡＭの大きさ	ＳＷ３にセットする数値（１６進数）
１２８ＫＢ	００
２５６ＫＢ	０１
３８４ＫＢ	０２
５１２ＫＢ	０３
６４０ＫＢ	０４

《ＰＣ－８８０１》
　システムの構成に合わせて、本体後面のディップスイッチとジャンパースイッチをセットする。ディップスイッチは、上がＯＮ。ＳＷ１の１と２はＯＦＦ、３と４はＯＮ（８０字／行、２５行／画面）、５と６はＯＦＦ。ＳＷ２は通信機能の制御に使うが、全てＯＦＦ。高解像度ディスプレーを使うときは、ジャンパースイッチのＨにソケットをさす。白黒ディスプレーを使うときは、Ｍにソケットをさす（カラーのときはどこでもよい「ＰＣ－８８０１のジャンパースイッチ」参照）。ＲＳ－２３２Ｃ（通信機能）の二組のスイッチは、さわらない。ここまでセットしたら、ＰＣ－８８０１以外の周辺装置から順次電源を入れ、最後にＰＣ－８８０１の電源を入れる。Ｈｏｗ　ｍａｎｙ　ｆｉｌｅｓ（０－１５）？という表示（注）がでたら、ＲＥＴキーを押す。表示されないときは、電源を切り、装置の接続、ディップスイッチ、ジャンパースイッチをチェックすること。



ディップスイッチ　　　　ジャンパースイッチ

　（注）Ｈｏｗ　ｍａｎｙ　ｆｉｌｅｓ（０－１５）？
　このとき入力する数値で、ＢＡＳＩＣで同時に使うファイル番号の最大数を決める。ファイルを扱うプログラムのときに必要。この値はシステム立ち上げ時に決まる。変更するときはリセット（本体前面左のリセットボタンを押す）した後で再入力する。

☆システムのチェックと時計・カレンダーのセット☆

《システムのチェック》
　ＭＯＮと入力してＲＥＴキーを押してモニターモードにする。次にＴＭと入力してＲＥＴキーを押すと、２０分程度でシステムのメモリーチェックが終る。正常に終われば、Ｔｅｓｔ　Ｃｏｍｐｌｅｔｅ！　と表示する。（ＰＣ－８８０１では、Ｔｅｓｔ　ｃｏｍｐｌｅｔｅ）。その後、リセットしてＢＡＳＩＣモードに戻す。
《ＴＩＭＥ＄・ＤＡＴＥ＄のセット》
　ＰＣ－９８０１・ＰＣ－８８０１は、ＴＩＭＥ＄（時計）とＤＡＴＥ＄（カレンダー）を持っている。ＴＩＭＥ＄は時分秒（00：00：00～23：59：59）を”ＨＨ：ＭＭ：ＳＳ”で持ち、ＤＡＴＥ＄は年月日を”ＹＹ／ＭＭ／ＤＤ”でもつ。一度合わせると内蔵電池の働きで、電源を切っても正しい年月日（（８８）は月日のみ）と時刻を示す。年だけは、年が変わったときに変更する必要がある。
　たとえば　１９８３年1月1日午後1時1分1秒をセットするには、次のようにする。
ＴＩＭＥ＄＝”１３：０１：０１”　と入力しＲＥＴキーを押す。
ＤＡＴＥ＄＝”８３／０１／０１”　と入力しＲＥＴキーを押す。

☆ＤＩＳＫ－ＢＡＳＩＣを使う場合☆
　ディスク装置の電源スイッチを入れた後、システムディスク（ＤＩＳＫ－ＢＡＳＩＣの入ったディスク）をドライブ１に入れる。ＰＣ本体の電源を入れると、システムディスクを読み込み、ＤＩＳＫ－ＢＡＳＩＣがスタートする。ＰＣ－９８８１（8インチ標準ディスク装置）は　本体と電源スイッチが共通なので、本体のスイッチ投入後、システムディスクをセットし、リセットする。8インチ標準フロッピーディスクとミニフロッピーディスクを接続しているときは8インチ標準フロッピーディスクが優先される。

☆ストップキー、リセットボタン☆
　プログラムを実行中に中止したいときや、リストを途中で止めたいときに、ＳＴＯＰキーを押す。ＳＴＯＰキーを押しても、その他のキーを押しても反応が無くなってしまったときには、リセットする。リセットするとシステムは電源を入れた直後の状態に戻る。それまで入力したプログラムなどは消えてしまう。ＳＴＯＰキーを押しながらリセットボタンを押すと、プログラムを消さないでリセットできる（ウォームスタート）。

☆ＰＣ－９８０１のディップスイッチ・メモリースイッチ☆

ディップスイッチとメモリースイッチの機能は次のとおり。

《ＰＣ－９８０１のディップスイッチ》

スイッチ	スイッチ番号	設定項目	ＯＮにする	ＯＦＦにする
ＳＷ１	1	使用ディスプレー	専用高解像度ディスプレー	専用高解像度以外のディスプレー
	2 3 4 5 6	モノクロディスプレーに表示する画面	グラフィック第１画面だけ グラフィック第２画面だけ グラフィック第３画面だけ テキスト画面だけ テキスト画面とグラフィックの全画面	どれか一つだけＯＮにする その他はＯＦＦ
	7 8 9 10	ＲＳ－２３２ＣのクロックをＰＣ－９８０１タイマーかモデムのどちらからとるかきめる	ＰＣ－９８０１のタイマー モデム ＰＣ－９８０１のタイマー モデム	モデム ＰＣ－９８０１のタイマー モデム ＰＣ－９８０１のタイマー
ＳＷ２	1	使用システム	つねにＯＦＦ	Ｎ８８－ＢＡＳＩＣ
	2	モード	ターミナルモード	ＢＡＳＩＣモード
	3 4	テキスト画面の文字数	８０文字／行 ２５行／画面	４０文字／行 ２０行／画面
	5	初期化時のメモリースイッチの状態	メモリースイッチの状態を変化させない	メモリースイッチをシステム既定値で初期化
	6～8	使わない		

《メモリースイッチ》
ＰＣ－９８０１本体の不揮発メモリーを用いたスイッチ。８ビットが一つのスイッチに対応している。モニターのＳＳＷ（セットスイッチ）でセットする。ＳＷ４～ＳＷ７は、未使用。

スイッチ	ＳＷ１	ＳＷ２	ＳＷ３	ＳＷ４	ＳＷ５	ＳＷ６	ＳＷ７
アドレス	A3FE2	A3FE6	A3FEA	A3FEE	A3FF2	A3FF6	A3FFA

ＳＷ１、ＳＷ２はＲＳ－２３２Ｃに使う。「通信機能」参照。ＳＷ４からＳＷ７は、未使用。メモリースイッチは、ディップスイッチＳＷ２の５をＯＦＦにすると電源を入れたり、リセットしたときにシステム既定値（ＳＷ１は４８Ｈ、ＳＷ２は０５Ｈ、ＳＷ３は００Ｈ）がセットされる。ＳＷ３の機能は次のとおり。

スイッチ	設定項目	機能	ビット位置 7	6	5	4	3	2	1	0
ＳＷ３	メモリーサイズ	１２８Ｋバイト						0	0	0
		２５６Ｋバイト						0	0	1
		３８４Ｋバイト						0	1	0
		５１２Ｋバイト						0	1	1
		６４０Ｋバイト						1	0	0

スイッチ	設　定　項　目（つづき）	機　能	ビット位置							
			7	6	5	4	3	2	1	0
SW3	PC－9806 （数値データプロセッサー）	なし あり					0 1			
	ODA系プリンター使用コード	JIS8ビット JIS7ビット				0 1				
	プリンターインタフェースの型式	セントロニクス ODA			0 1					
	テキスト画面の色	白 緑		0 1						
	DELコードの扱い	BSコード NULコード	0 1							

☆PC－8801のディップスイッチ・ジャンパースイッチ☆

PC－8801のディップスイッチとジャンパースイッチの機能は次のとおり。

《PC－8801のディップスイッチ》

スイッチ	スイッチ番号	設　定　項　目	ONにする	OFFにする
SW1	1	使用システム	N－BASIC	N88－BASIC
	2	モード	ターミナルモード	BASICモード
	3	テキスト画面の文字数	80字／行	40字／行
	4		25行／画面	20行／画面
	5	Sパラメーター	有効	無効
	6	DELコードの処理	BSとする	NULとする
SW2	1	パリティーチェック	あり	なし
	2	パリティー指定	偶数パリティー	奇数パリティー
	3	データビット長	8ビット	7ビット
	4	ストップビット	2	1
	5	Xパラメーター	有効	無効
	6	通信方式	半二重	全二重

（注）SW1の5から6、SW2の設定項目はターミナルモードの初期設定

《PC－8801のジャンパースイッチ》

（ブロック名）RS－232C	1	2	3	4	5	6	7	8	I	E
＜伝送速度＞										
75ボー	○									
150ボー		○								
300ボー			○							
600ボー				○						
1200ボー					○					
2400ボー						○				
4800ボー							○			
9600ボー								○		
＜RS－232Cのクロック＞										
PC－8801のタイマー									○	
モデム										○

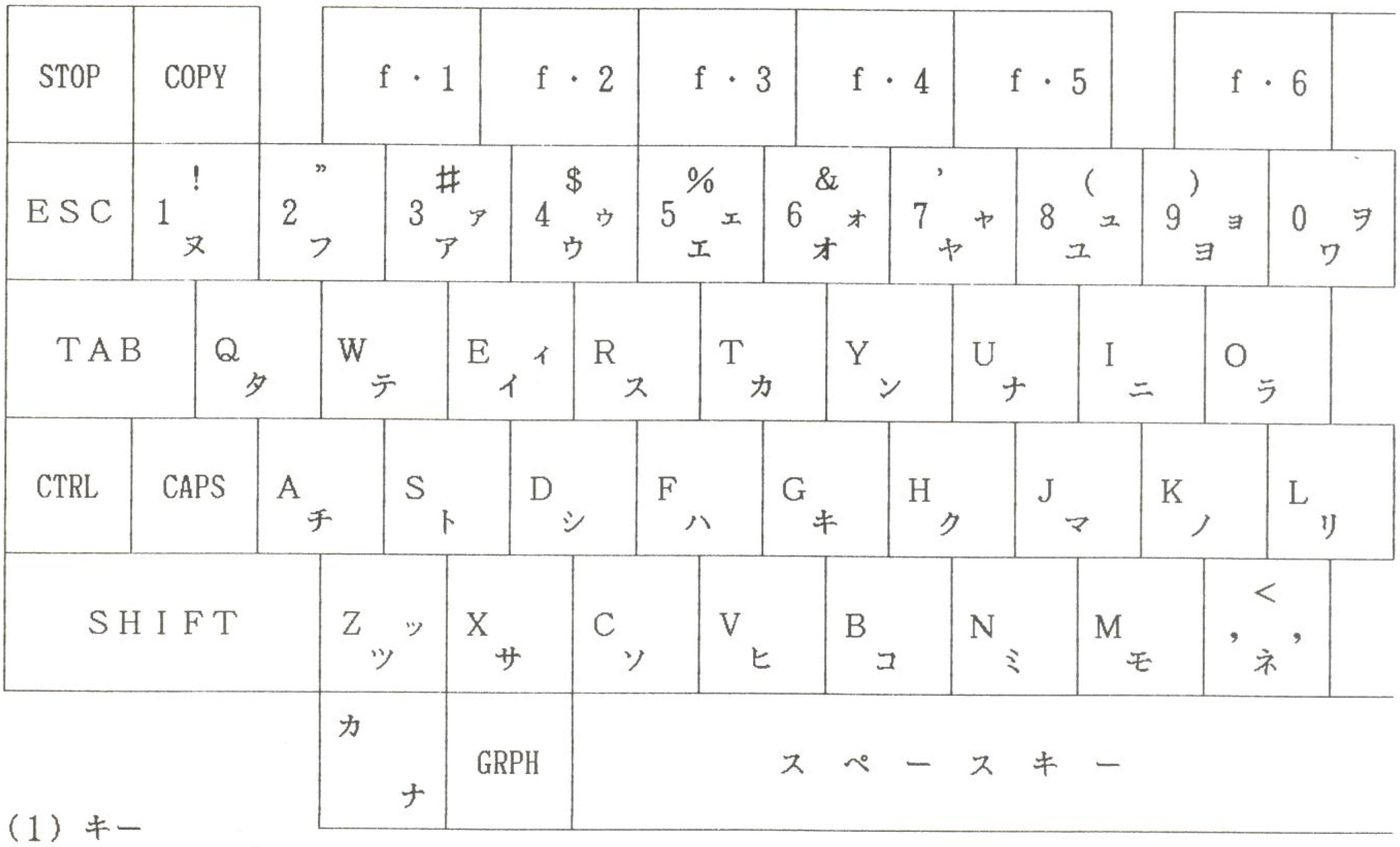
（スイッチ番号）

（ブロック名）CRT	S	H	M	T	0	1	2
＜使用ディスプレー＞							
専用高解像度以外のディスプレー	○						
専用高解像度ディスプレー		○					
＜モノクロ出力画面＞							
テキスト画面とグラフィック画面			○				
テキスト画面だけ				○			
グラフィック第1画面だけ					○		
グラフィック第2画面だけ						○	
グラフィック第3画面だけ							○

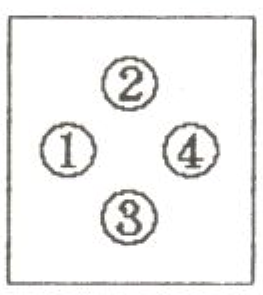
（表の見方）○印の左の項目を設定する場合、○印の上のスイッチ番号にソケットをさしこむ

キー配置図　PC-9801

ファンクションキー　f1→load"　f2→auto　f3→go to　f4→list　f5→run+RET
の定義 ——→　f6→save"　f7→key　f8→print　f9→edit.+RET　f10 →cont+RET

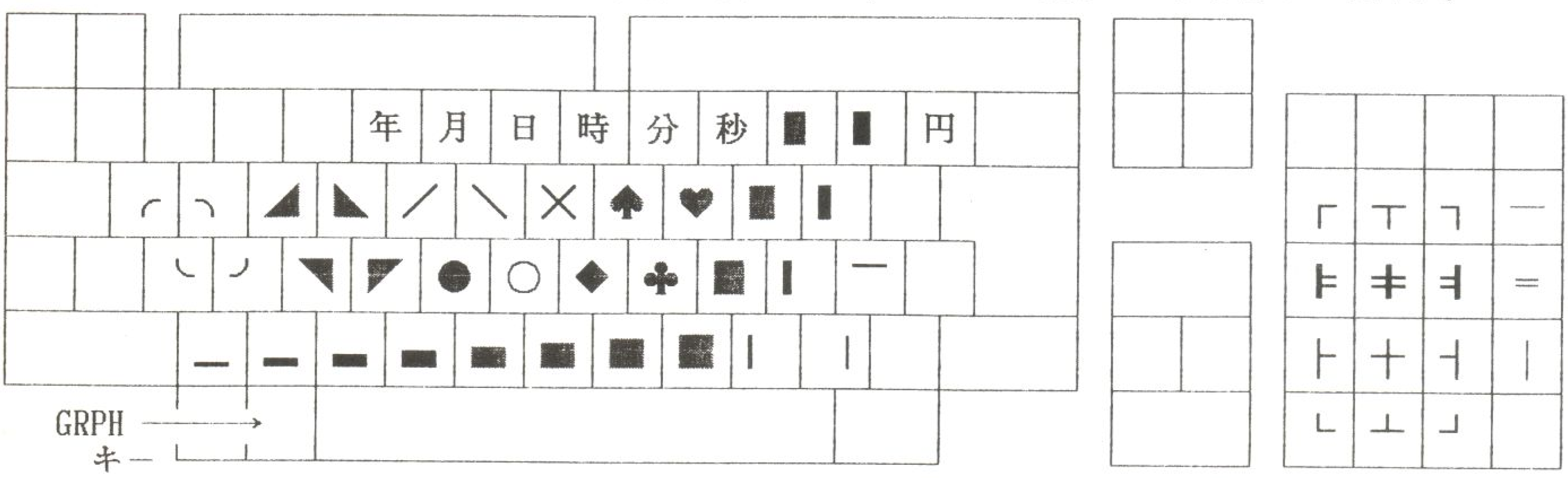


(1) キー



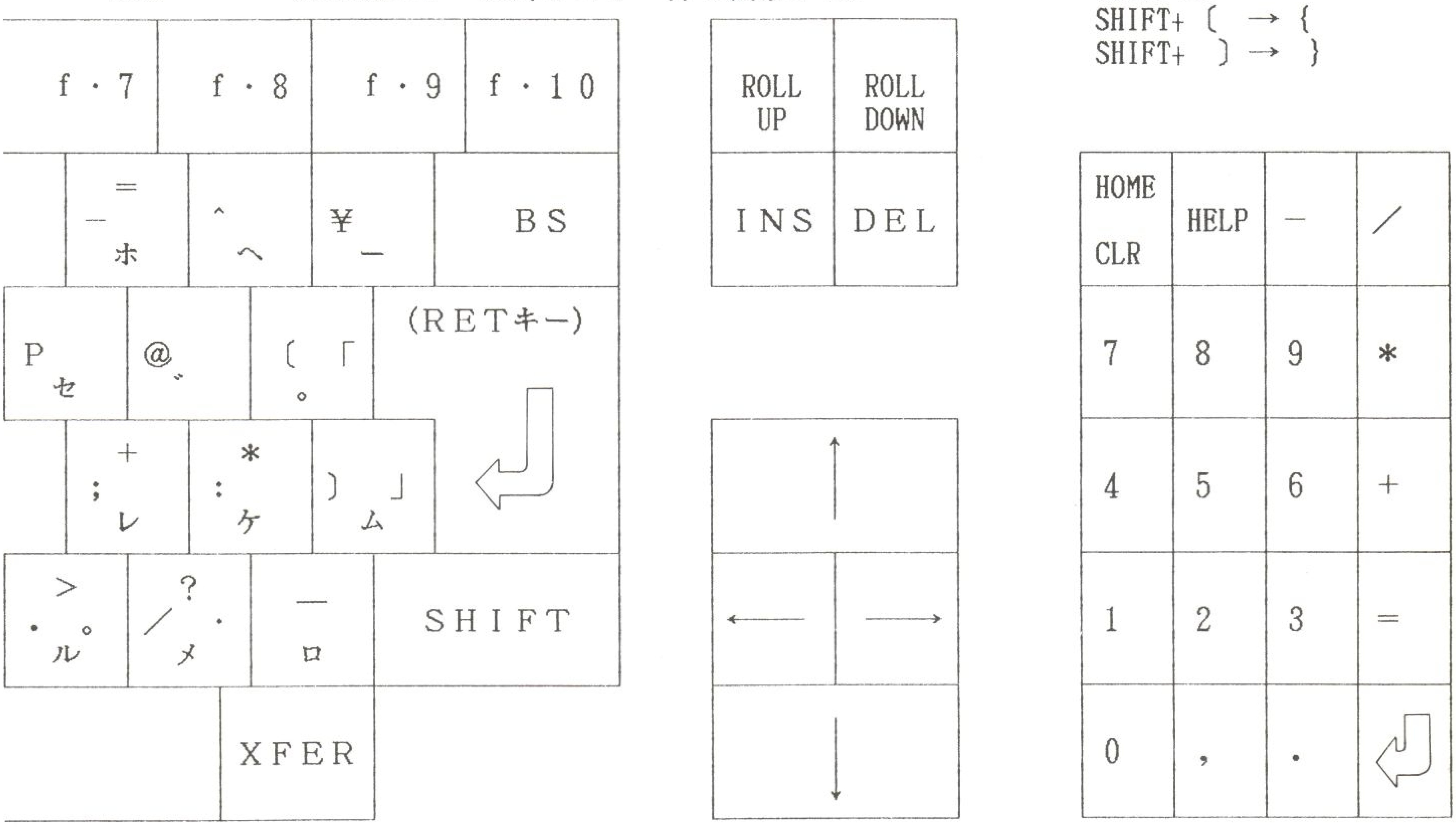
① キーを押すと①の位置の文字が出る。英字の場合は小文字。
② SHIFT を押してキーを押すと②の位置の文字が出る。
英字の場合は大文字が出る。
③ カナキーを押してキーを押すと③の位置の文字が出る。
カナキーとSHIFT を押してキーを押すと④の位置の文字が出る。

(2) COPYキー　グラフィックとテキストの混在した画面をプリンターに出力する。
テキスト画面の内容だけをプリンターに出力するときは、CTRL＋COPY。
グラフィック画面の内容だけをプリンターに出力するときは、GRPH＋COPY。
(3) STOPキー　プログラムの強制停止。再開はＣＯＮＴコマンド（ＣＯＮＴ参照）
ターミナルモードから戻るときは SHIFT+STOP 。
(4) ＥＳＣキー　ターミナルモードのときＥＳＣコードを入力する。
(5) ＴＡＢキー　カーソルを８文字ごとのタブ位置へ移す。タブ位置までは空白になる。
(6) ＣＡＰＳキー　押して英字のキーを押すと英大文字。解除はもう一度押す。
(7) カナキー　押すとカナモード。解除はもう一度押す。
(8) スペースキー　押すと空白をひとつとる。
(9) ＧＲＰＨキー　下の文字および図形を出すには，GRPHキーを押しながら各キーを押す。



（１０）ROLL UP　　テキスト画面を上へスクロールする。
　　　　ROLL DOWN　テキスト画面を下へスクロールする。
（１１）HOME　　　SHIFT+CLR カーソルが左上へ。
　　　　CLR　　　　押すと画面クリアー。
　　　　HELP　　　SYNTAXエラー以外のエラー行を表示する。

キーボードにない文字
SHIFT+￥　→　¦
SHIFT+@　→　～
SHIFT+〔　→　{
SHIFT+〕　→　}



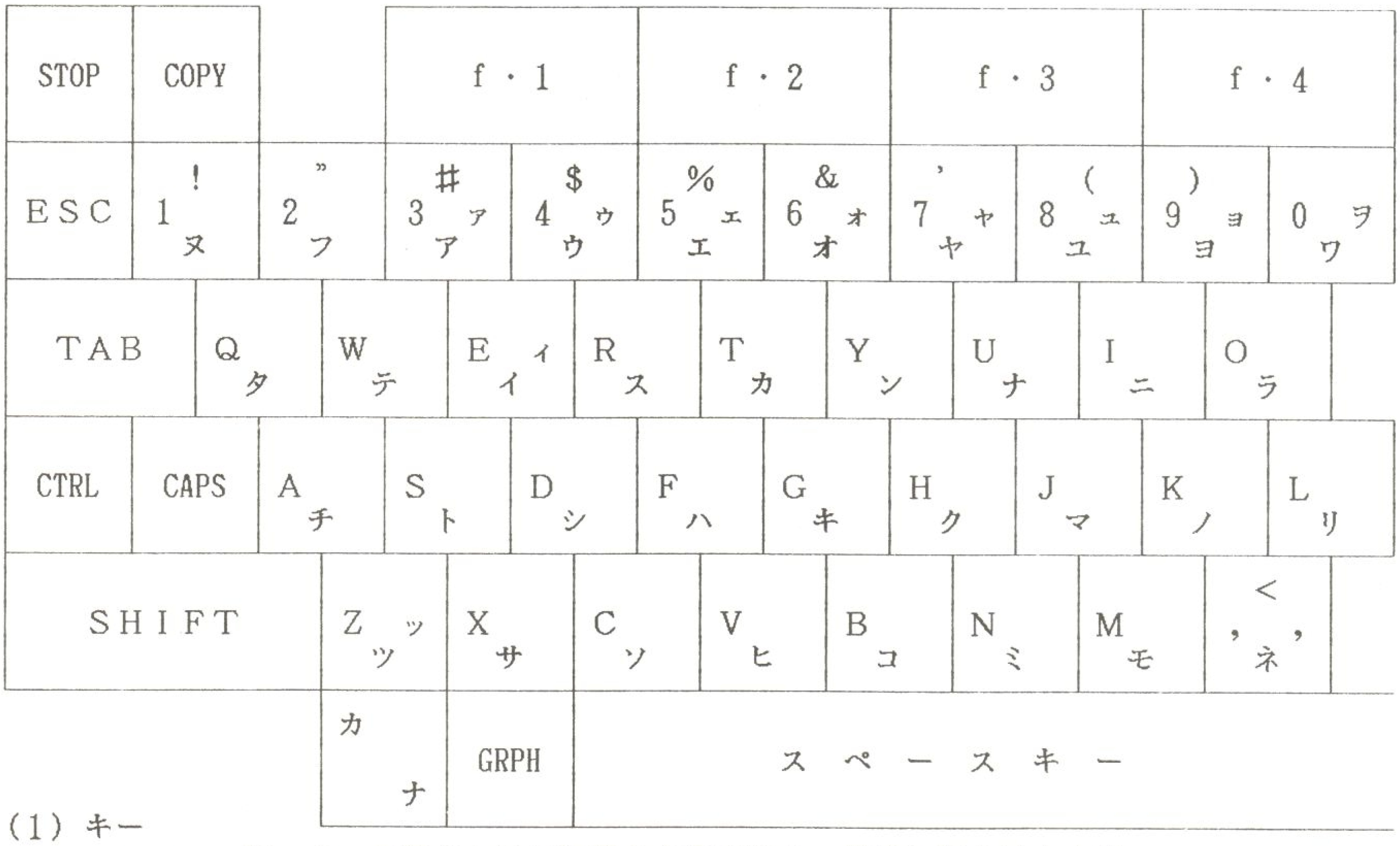
（１２）ＩＮＳキー　　挿入モードにする。解消は再度押すか、カーソルの移動でする。
（１３）ＤＥＬキー　　カーソルの左１文字を消してつめる。
（１４）ＲＥＴキー　　入力の終わりに必ず押すこと。
（１５）矢印　　　　　カーソルを各方向に移動。
（１６）ＢＳキー　　　カーソルの左１文字を消してつめる。ターミナルモードのときＢＳコードを入力。
（１７）ＸＦＥＲキー　ＣＴＲＬ＋ＸＦＥＲで日本語モード。解除はもう１度ＣＴＲＬ＋ＸＦＥＲ。
（１８）ＣＴＲＬキー　スクリーンエディットのときに使う。このキーを押しながら下表のキーを押す。

N88-BASICモード	Ａ→ＨＥＬＰキーと同じ Ｂ→前の単語にカーソルを移す Ｃ→ＳＴＯＰと同じ Ｄ→カーソル上の単語を消す Ｅ→カーソルから行の終わりまでを消す Ｆ→次の単語にカーソルを移す Ｇ→ＢＥＥＰ Ｈ→ＤＥＬキーと同じ Ｉ→ＴＡＢキーと同じ Ｊ→ラインフィードを作る。ＩＮＳキーを押してこのキーを押すと、カーソル以後を次の行に分割する	Ｋ→ＨＯＭＥキーと同じ Ｌ→ＣＬＲキーと同じ Ｍ→ＲＥＴキーと同じ Ｏ→実行中、テキスト画面への表示を消す　解除はもう一度押す Ｒ→ＩＮＳキーと同じ Ｓ→プログラムの一時停止。再開はＣＴＲＬ・ＳＨＩＦＴ・カナ・ＧＲＰＨ・ＣＡＰＳ・ＳＴＯＰ・ＣＯＰＹ・ロ・ＸＦＥＲ以外のキー Ｕ→カーソルの行を全部消す Ｘ→カーソルを行の最後に移す
モニターモード	Ａ→モニター・コマンドの形式を画面に出力。ＨＥＬＰキーと同じ Ｂ→Ｓコマンド以外のときＢＡＳＩＣモードに戻す Ｃ→Ａ・Ｄ・Ｌコマンドの強制停止 Ｄ→ディスクの内容を見る（セクター単位で指定する） Ｒ→ディスクのデータを読む（アドレス単位で指定する） Ｓ→Ｄ・Ｌコマンドの一時停止　再開はＳＴＯＰ・ＣＴＲＬ＋Ｃ以外のキー Ｗ→ディスクへデータを書く（アドレス単位で指定する）	

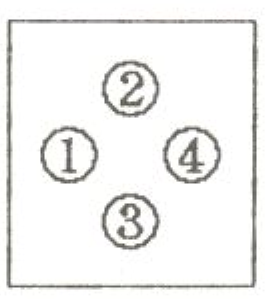
キー配置図　PC-8801

ファンクションキーの定義 ──→

SHIFT+f1→save"	SHIFT+f2→key	SHIFT+f3→print	SHIFT+f4→edit.RET
f1のみ──→load"	f2のみ→auto	f3のみ──→go to	f4のみ──────→list



(1) キー



① キーを押すと①の位置の文字が出る。英字の場合は小文字。
② SHIFT を押してキーを押すと②の位置の文字が出る。
　英字の場合は大文字が出る。
③ カナキーを押してキーを押すと③の位置の文字が出る。
　カナキーとSHIFT を押してキーを押すと④の位置の文字が出る。

(2) COPYキー　グラフィックとテキストの混在した画面をプリンターに出力する。
　テキスト画面の内容だけをプリンターに出力するときは、CTRL＋COPY。
　グラフィック画面の内容だけをプリンターに出力するときは、GRPH＋COPY。

(3) STOPキー　プログラムの強制停止。再開はCONTコマンド（CONT参照）
　ターミナルモードから戻るときは SHIFT+STOP 。

(4) ESCキー　ターミナルモードのときESCコードを入力する。

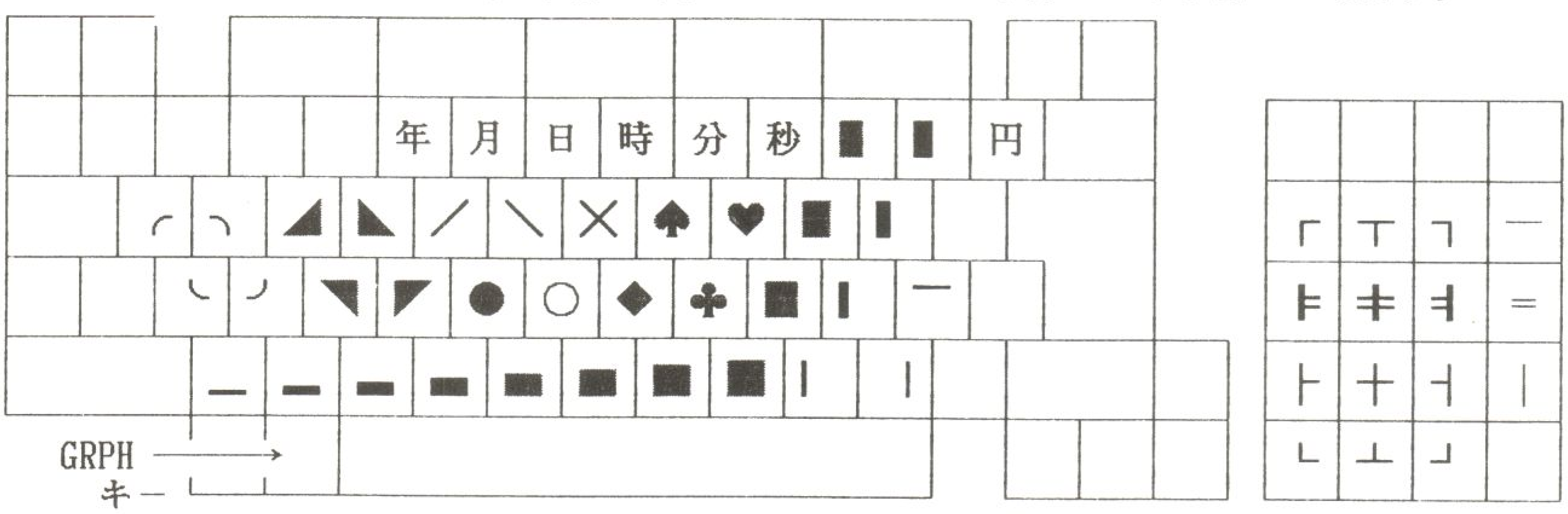
(5) TABキー　カーソルを８文字ごとのタブ位置へ移す。タブ位置までは空白になる。

(6) CAPSキー　押して英字のキーを押すと英大文字。解除はもう一度押す。

(7) カナキー　押すとカナモード。解除はもう一度押す。

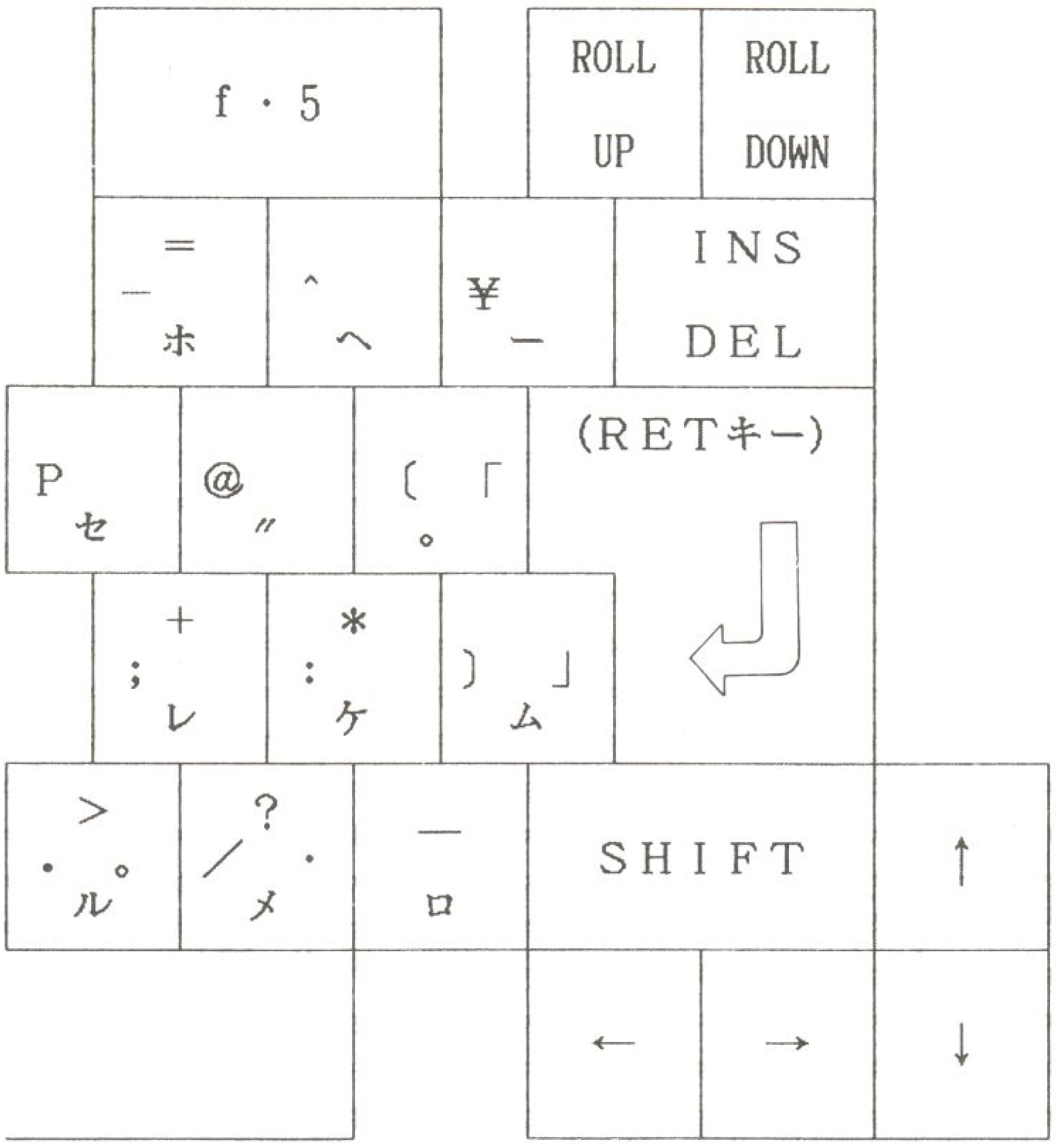
(8) スペースキー　押すと空白をひとつとる。

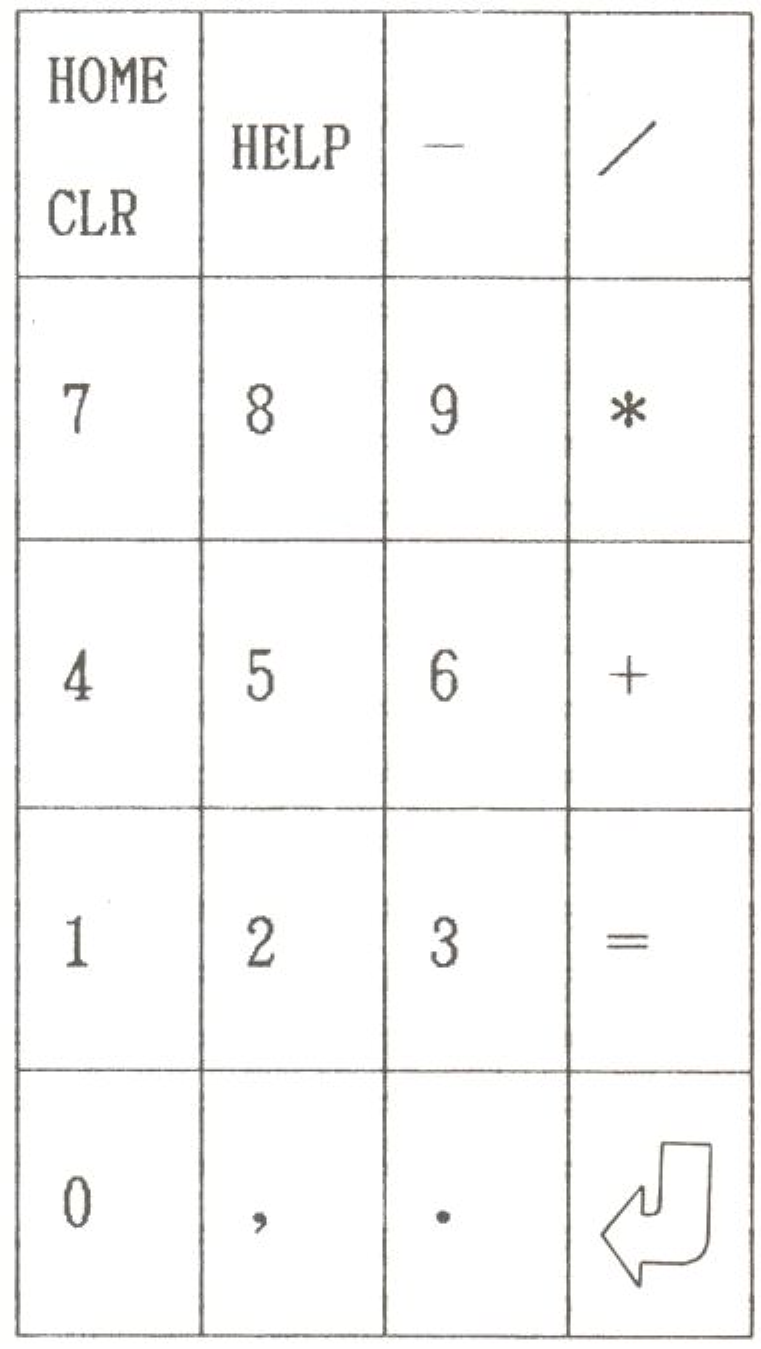
(9) GRPHキー　下の文字および図形を出すには、GRPHキーを押しながら各キーを押す。



（１０）ROLL UP　テキスト画面を上へスクロールする。
　　　　ROLL DOWN テキスト画面を下へスクロールする。
（１１）HOME　　SHIFT+CLR カーソルが左上へ。
　　　　CLR　　　押すと画面クリアー。
　　　　HELP　　SYNTAXエラー以外のエラー行を表示する。
f5のみ──→run+RET　SHIFT+f5→cont+RET

キーボードにない文字
SHIFT+¥ → ¦
SHIFT+@ → ～
SHIFT+〔 → {
SHIFT+〕 → }





（１２）ＩＮＳキー　　挿入モードにする。解消は再度押すか、カーソルの移動でする。
　　　　ＤＥＬキー　　カーソルの左１文字を消してひとつつめる。
（１３）ＲＥＴキー　　入力の終わりに必ず押すこと。
（１４）矢印　　　　　カーソルを各方向に移動。

（１５）ＣＴＲＬキー　スクリーンエディットのときに使う。このキーを押しながら下表のキーを押す。

Ｎ８８－ＢＡＳＩＣモード	Ａ→ＨＥＬＰキーと同じ Ｂ→前の単語にカーソルを移す Ｃ→ＳＴＯＰと同じ Ｄ→カーソル上の単語を消す Ｅ→カーソルから行の終わりまでを消す Ｆ→次の単語にカーソルを移す Ｇ→ＢＥＥＰ Ｈ→ＤＥＬキーと同じ Ｉ→ＴＡＢキーと同じ Ｊ→ラインフィードを作る。ＩＮＳキーを押してこのキーを押すと、カーソル以後を次の行に分割する	Ｋ→ＨＯＭＥキーと同じ Ｌ→ＣＬＲキーと同じ Ｍ→ＲＥＴキーと同じ Ｏ→実行中、テキスト画面への表示を消す　解除はもう一度押す Ｒ→ＩＮＳキーと同じ Ｓ→プログラムの一時停止。再開はＣＴＲＬ・ＳＨＩＦＴ・カナ・ＧＲＰＨ・ＣＡＰＳ・ＳＴＯＰ・ＣＯＰＹ・ロ以外のキー Ｕ→カーソルの行を全部消す Ｘ→カーソルを行の最後に移す
モニターモード	Ａ→モニター・コマンドの形式を画面に出力。ＨＥＬＰキーと同じ Ｂ→Ｓコマンド以外のときＢＡＳＩＣモードに戻す Ｃ→Ａ・Ｄ・Ｌコマンドの強制停止 Ｄ→ディスクの内容を見る。（セクター単位で指定する） Ｒ→ディスクのデータを読む。（アドレス単位で指定する） Ｓ→Ｄ・Ｌコマンドの一時停止。再開はＳＴＯＰ・ＣＴＲＬ＋Ｃ以外のキー Ｗ→ディスクへデータを書く。（アドレス単位で指定する）	

Ⅰ 基礎編 　1. BASICの基本文法と機能

☆ＢＡＳＩＣの基本文法☆

（1）ＢＡＳＩＣ言語で書かれた１つ１つの処理手順を文（ステートメント）という。文の組み合わせでプログラムを作る。ＢＡＳＩＣへの指示を命令（コマンド）という。命令の中には文として働くものもある。

（2）プログラムは小文字で入力しても大文字で入力してもかまわない。

（3）ダイレクトモードでは電卓のように使える。この場合、行番号をつけない。

```
PRINT 3+4
? SQR(2)
FOR J=1 TO 10:PRINT J*5:NEXT J
```

プログラムモードでは、まずプログラムとして記憶する。それから実行する。したがって、何回でも実行できる。行の始めに必ず行番号をつける。

```
10 INPUT R
20 A=2*3.14159*R
30 PRINT A
40 END
```

（4）行番号は１から６５５２９までの正整数でつける。１行の長さは２５５文字まで。行番号や空白も１字に数える。コロン（:）で区切ると１行に複数の文が書ける。これをマルチステートメントという。

（5）定数とはＢＡＳＩＣプログラムの実行時に、そのまま使える値をいう。数値定数と文字定数とがある。文字定数は引用符（”）で囲む。

- １０、１．５、－１００ …… 数値定数
- ”ＡＢＣ”、”イロハ” …… 文字定数

（6）変数とはＢＡＳＩＣプログラムのなかで使う値を表す名前をいう。

- 変数
  - 文字変数
  - 数値変数
    - 整数型変数
    - 実数型変数
      - 単精度型
      - 倍精度型

変数名は最大４０文字の英数字とピリオド（．）で表す。ただし最初の１文字は英字とする。大文字と小文字の区別はない。変数名はＢＡＳＩＣの命令や文と同一であってはならない。また、ＦＮやＵＳＲで始まる変数名は使えない。

- ＰＲＩＮＴ …… 変数名として使えない。
- ＰＲＩＮＴＸ、ＺＰＲＩＮＴ …… 変数名として使える。

実数を整数型変数に代入すると小数点以下が四捨五入される。

（7）式とは定数、変数、関数、演算子の組み合わせで、結果が数値か文字列になるものをいう。

（8）演算は算術演算、関係演算、論理演算、関数、文字列演算がある。「文字と演算子」参照。

算術演算 ……… 算術演算子（＋や－など）を使って四則演算やべき乗（^）計算をする。
関係演算 ……… 関係演算子（＜や＝など）を使って数値や文字列の比較をする。結果は真（－１）偽（０）で得られる。
論理演算 ……… 論理演算子（ＡＮＤやＯＲなど）を使ってビット単位に論理演算をする。論理演算子を使って複数の条件を判定できる。
関数 ……………… 与えた引数を使って一定の演算をし、結果を返す。ＢＡＳＩＣでは組み込み関数のほかに、ユーザーが定義するユーザー関数が使える。
文字列演算 …… 文字列を演算子（＋）で連結することができる。
ＰＲＩＮＴ　”ＡＢＣ”＋”イロハ”　を実行すると　ＡＢＣイロハ　となる。

演算の優先順位は、
カッコで囲まれた式 →関数 →べき乗 →負号 →＊，／ →¥ →ＭＯＤ →＋，－ →
関係演算子 →ＮＯＴ →ＡＮＤ →ＯＲ ─┬→ＸＯＲ →ＩＭＰ →ＥＱＶ　（９８）
　　　　　　　　　　　　　　　　　　└→ＸＯＲ，ＥＱＶ →ＩＭＰ　（８８）
同順位であれば、左から右へと演算する。

（9）ラベル
プログラムの流れを変えるとき行番号のかわりにラベルを使える。ラベルを使うときは以下の点に注意すること。

・ラベル名の先頭はアスタリスク（＊）で、＊の次は英字
・＊英字　のあとの文字は、英数字でよい。ラベル名は最大２５５文字まで
・ラベル名はＢＡＳＩＣの命令や文と同一であってはならない。また、ＦＮやＵＳＲで始まるラベル名は使えない
・英字は小文字を使っても大文字になる
・とび先のラベル名は行の最初に置く
・ラベル名と同じ行に命令を書くときは、コロン（：）またはスペースで区切る
・ラベル名はＬＩＳＴやＤＥＬＥＴＥなど、コマンドの行番号としても使える

（例）
```
10  *LAB1
20  PRINT "ABC"
30  GOTO *LAB1     （これで行番号10へもどる）
```

（１０）ファイルディスクリプター（ファイル記述子）は、”デバイス名：ファイル名”で表す（例１）。デバイス名を省略するとＲＯＭ－ＢＡＳＩＣであればカセットテープになり、ＤＩＳＫ－ＢＡＳＩＣであれば「１：」になる（例２）。
また、”　”で囲んだ文字列だけでなく文字式も書ける（例３）。

（例１）　”ＫＹＢＤ：ＦＩＬＥ１”　キーボードから読み込む
（例２）　”ＡＢＣ”　デバイス名を省略した形で、”１：ＡＢＣ”と同じ
（例３）　Ａ＄＝”ＦＩＬＥ２”
　　　　　ＬＯＡＤ　”２：”＋Ａ＄

（１１）ファイルディスクリプターで使うデバイス名は次のように指定する。

キーボード	ＫＹＢＤ：　注１
スクリーン	ＳＣＲＮ：　注１
プリンター	ＬＰＴ１：　注１

カセットテープ	ＣＡＳ１～２：　注１
ディスク１～１０	１～１０：　注２
ＲＳ－２３２Ｃポート	ＣＯＭ１～３：

注１：ファイル名を省略できる。　注２：（８８）では８までしかとれない。

「ＣＯＭ：」はファイル名のかわりにオプションを指定する。「通信機能」参照。

☆ＢＡＳＩＣの機能☆

《割り込み処理》

「割り込み」とは、プログラムを実行中に割り込みの原因が発生（ＨＥＬＰキーを押したなど）すると実行していたプログラムを中断し、割り込みの原因に対応した処理をしたのちに、中断していたプログラムを再開する一連の処理をいう。

N88－ＢＡＳＩＣでは、ファンクションキー、ストップキー、ヘルプキー、ＲＳ－２３２Ｃのポート、ライトペン、リアルタイム・タイマーからの割り込み処理ができる。

〔ＯＮ　割り込みの種類　　ＧＯＳＵＢ　割り込みルーチンの開始行〕
　割り込みが発生したときに実行するサブルーチンを指定する。
〔割り込みの種類　ＯＮ〕
　割り込みの要求を許可する。割り込みが発生すると、指定した割り込みルーチンへ処理が移る。
〔割り込みの種類　ＯＦＦ〕
　割り込みを禁止する。この場合、割り込みの原因が発生しても割り込み処理はしない。
〔割り込みの種類　ＳＴＯＰ〕
　割り込みを中断する。中断している間に発生した割り込みは、その時点では処理せずに覚えておく。〔割り込みの種類　ＯＮ〕とすると、覚えておいた割り込み処理をする。

割り込みが同時に発生したときは、割り込みに優先順位をつけて処理する。優先順位は、ストップキー、ヘルプキー、リアルタイム・タイマー、ファンクションキー１～１０、ＲＳ－２３２Ｃの１～３、ライトペンの順である。ただし、割り込みルーチンの処理中に他の割り込みが発生すると、優先順位に関係なくその割り込みを受けつける。たとえば、ストップキーの割り込み処理中にヘルプキーを押すと、ヘルプキーの割り込みルーチンを先に処理したあとで、ストップキーの割り込み処理をする。

（例）ＨＥＬＰキーの割り込み処理

```
 10 ON HELP GOSUB *WARIKOMI
 20 HELP ON
 30 FOR J=1 TO 800
 40    PRINT J;
 50 NEXT J
 60 HELP OFF
 70 END
 80 *WARIKOMI
 90 PRINT "WARIKOMI"
100 RETURN
```

ＨＥＬＰキー割り込み処理の定義

```
ON HELP GOSUB *WARIKOMI
```

ＨＥＬＰキー割り込みの許可

```
HELP ON
```

メインのプログラムを実行

```
FOR J=1 TO 800

   PRINT J;

NEXT J
```

ＨＥＬＰキーを押す
↓
割り込みの発生
↓
→メインプログラムを中断 ……→ 割り込み処理ルーチン

```
*WARIKOMI
PRINT "WARIKOMI"
RETURN
```

←メインプログラムを再開 ←……

実行結果

```
1 2 3 WARIKOMI
4 5 6 ・・・・・・・・・
```

ＨＥＬＰキー割り込みの禁止

```
HELP OFF
```

《番地の指定方法》

（８８）は最大６４ＫＢのメモリーをもつ。番地は０Ｈ～ＦＦＦＦＨ（１０進数の０～６５５３５）の範囲で直接指定する。

（９８）は最大１ＭＢのメモリーをもつ。　メモリーを「セグメント」と呼ぶ６４ＫＢの連続したブロックに区切り、セグメントの先頭番地と、「オフセット」と呼ぶセグメント内の相対変位で番地を指定する。

オフセットもセグメントの先頭番地も０Ｈ～ＦＦＦＦＨ（１０進数の０～６５５３５）の範囲で指定するが、セグメントの先頭番地として指定した値は４ビット桁上げした値０Ｈ～ＦＦＦＦ０Ｈ（１０進数では１６倍して０～１，０４８，５６０）がセグメントの先頭番地になる。この値と、オフセットとして指定した値を加えた値が番地になる。本書ではセグメントの先頭番地として指定する値をセグメントベース番地またはセグメントベースと呼ぶ。

（例）セグメントベース番地を２１５１Ｈ、オフセットを１Ｃ７５Ｈとすると、物理番地は２３１８５Ｈ番地になる。

```
    ２１５１０Ｈ←………セグメントベース番地の指定を４ビット桁上げした値
＋   １Ｃ７５Ｈ←………オフセット
──────────
    ２３１８５Ｈ←………物理番地
```

セグメントベース番地を２１５１Ｈとすると、オフセットと合わせて、物理番地は２１５１０Ｈから３１５０ＦＨ番地まで表せる。

（９８）のＢＡＳＩＣでは、ＣＬＥＡＲ文とＤＥＦ　ＳＥＧ文でセグメントベース番地を指定する。ＢＬＯＡＤ、ＢＳＡＶＥ、ＣＡＬＬ、ＤＥＦ　ＵＳＲ、ＰＯＫＥ文とＰＥＥＫ、ＵＳＲ関数で、オフセットを指定する。ＶＡＲＰＴＲ文で変数があるセグメントベース番地とオフセットがわかる。

《日本語処理（ＰＣ－９８０１の場合）》

漢字ＲＯＭボード（ＰＣ－９８０１－０１）を実装すると、日本語処理ができる。日本語文字をプリンターに出力するには、ＰＣ－８８２２プリンター（ＰＣ－８８２１に漢字ＲＯＭボードを実装）を使う。

（９８）では漢字をそのまま文字列として扱える。

・日本語入力

キーボードからＪＩＳコード（１６進数）で入力する。日本語入力モードへの切り替えは【ＣＴＲＬ】＋【ＸＦＥＲ】。解除は再度【ＣＴＲＬ】＋【ＸＦＥＲ】。

・日本語文字の種類

文字の種類	テキスト画面	グラフィック画面	プリンター（ＰＣ－８８２２）
ＪＩＳ漢字	可	可	可
ＪＩＳ非漢字	可	可	可
半角文字	可（一部不可）	可	不可
１／４角文字	不可	可	不可
ケイ線文字	可	可	不可

可：表示または出力ができる。文字は漢字コード表参照

・日本語文字列の形式

日本語文字は１文字を２バイトで表す。（英数カナ文字はキャラクターコード表のとおり１文字１バイト）日本語文字列の最初と最後には２バイトずつのシフト記号をつけて英数カナ文字と区別する。

１６進コード	３１	３２	３３	１Ｂ	４Ｂ	４４	２Ｂ	４６	７Ｃ	１Ｂ	４８	４１	４２
文字列	１	２	３	ＫＩ		朝		日		ＫＯ		Ａ	Ｂ
	英数カナ文字			漢字シフトイン記号		日本語文字				漢字シフトアウト記号		英数カナ文字	

└【ＣＴＲＬ】＋【ＸＦＥＲ】で自動的に入る┘

・表示選択入力

オプションの「Ｎ８８日本語ＢＡＳＩＣ」を使うと、漢字の読みを入れて変換する表示選択方式で漢字を入力できる。「表示選択方式による日本語入力」参照。

《漢字処理（ＰＣ－８８０１の場合）》

漢字ＲＯＭボード（ＰＣ－８８０１－０１）を実装すると、漢字処理ができる。漢字を出力するプリンターは、（９８）と同じ。

（８８）では漢字をコードで指定するので、直接文字列にはできない。

・漢字表示

ＰＵＴ文でグラフィック画面に表示する。（ＰＵＴ文参照）

・漢字の種類

ＪＩＳ漢字、ＪＩＳ非漢字、半角文字、１／４角文字 ……… 漢字コード表参照

・（９８）と（８８）の違い

（９８）と（８８）では漢字の扱いが異なるので、プログラムは次のようにかわる。

（９８）の場合

```
１０　Ａ＄＝”朝”
２０　ＰＲＩＮＴ　Ａ＄
```

（８８）の場合

```
１０　Ａ＝＆Ｈ４４２Ｂ　………→「朝」の漢字コード
２０　ＰＵＴ（０，０），ＫＡＮＪＩ（Ａ）
```

《テキスト画面とグラフィック画面》

テキスト画面とグラフィック画面は独立している。表示はグラフィック画面にテキスト画面を重ねる。

・テキスト画面のモード

テキスト画面の文字数はＷＩＤＴＨ文で指定する。

文字数　×　行数	（９８）漢字の文字数
４０　×　２０	２０　×　２０
４０　×　２５	２０　×　２５
８０　×　２０	４０　×　２０
８０　×　２５	４０　×　２５

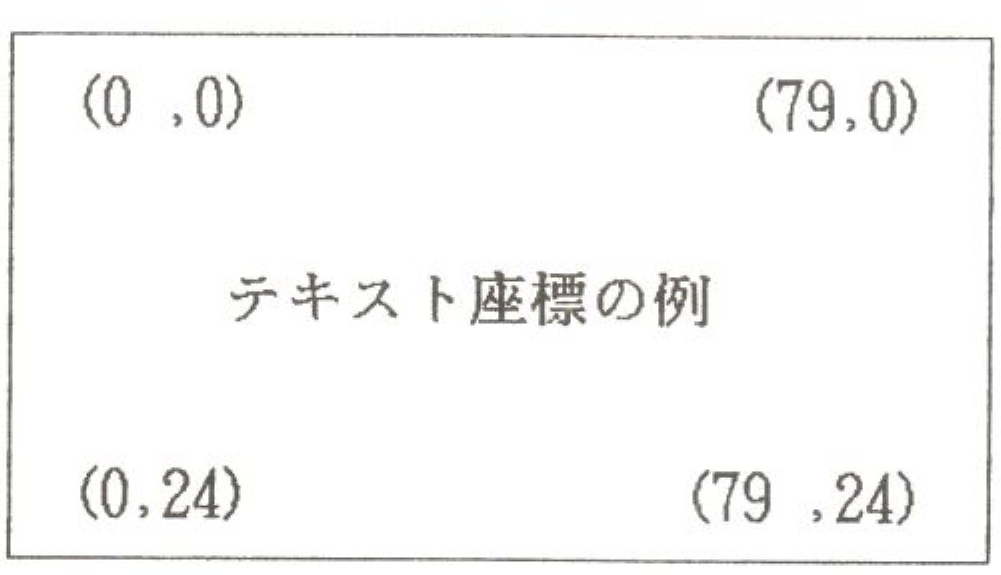
・グラフィック画面のモード

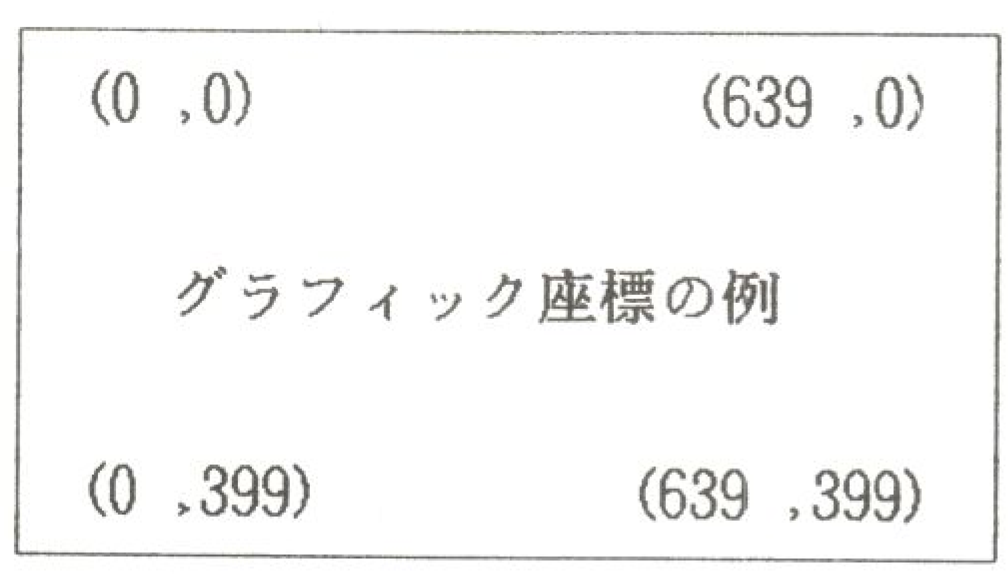
グラフィック画面のモードは、ＳＣＲＥＥＮ文で指定する。モードによってグラフィック画面を複数持てる。

ＰＣ－９８０１			ＰＣ－８８０１		
モード	ドット数	ページ数	モード	ドット数	ページ数
カラー	６４０×２００	２	カラー	６４０×２００	１
白黒	６４０×２００	６	白黒	６４０×２００	３
高分解白黒	６４０×４００	３	高分解白黒	６４０×４００	１
高分解カラー	６４０×４００	１			

《座標系》

テキスト画面の座標系をキャラクター座標、グラフィック画面の座標系をグラフィック座標という。



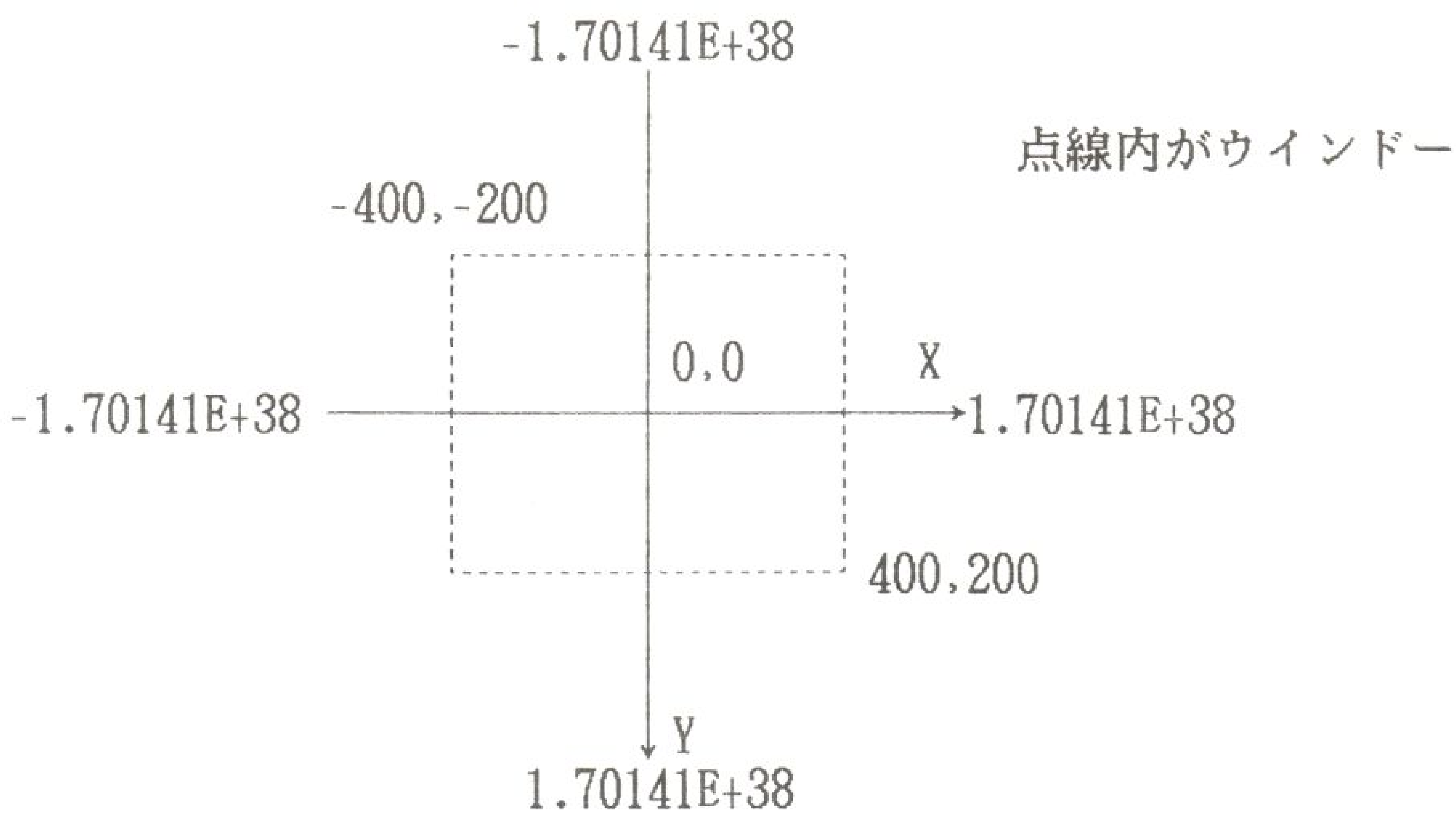


《グラフィック座標》
　グラフィック画面には６４０×４００の点（ドット）まで表現できる。ドットと１対１で対応した座標のほかに論理的な座標系によって多様な表現を可能にしている。
　グラフィックの座標系には、次の３種類ある。

・ワールド座標
・スクリーン座標
・オリジナルスクリーン座標

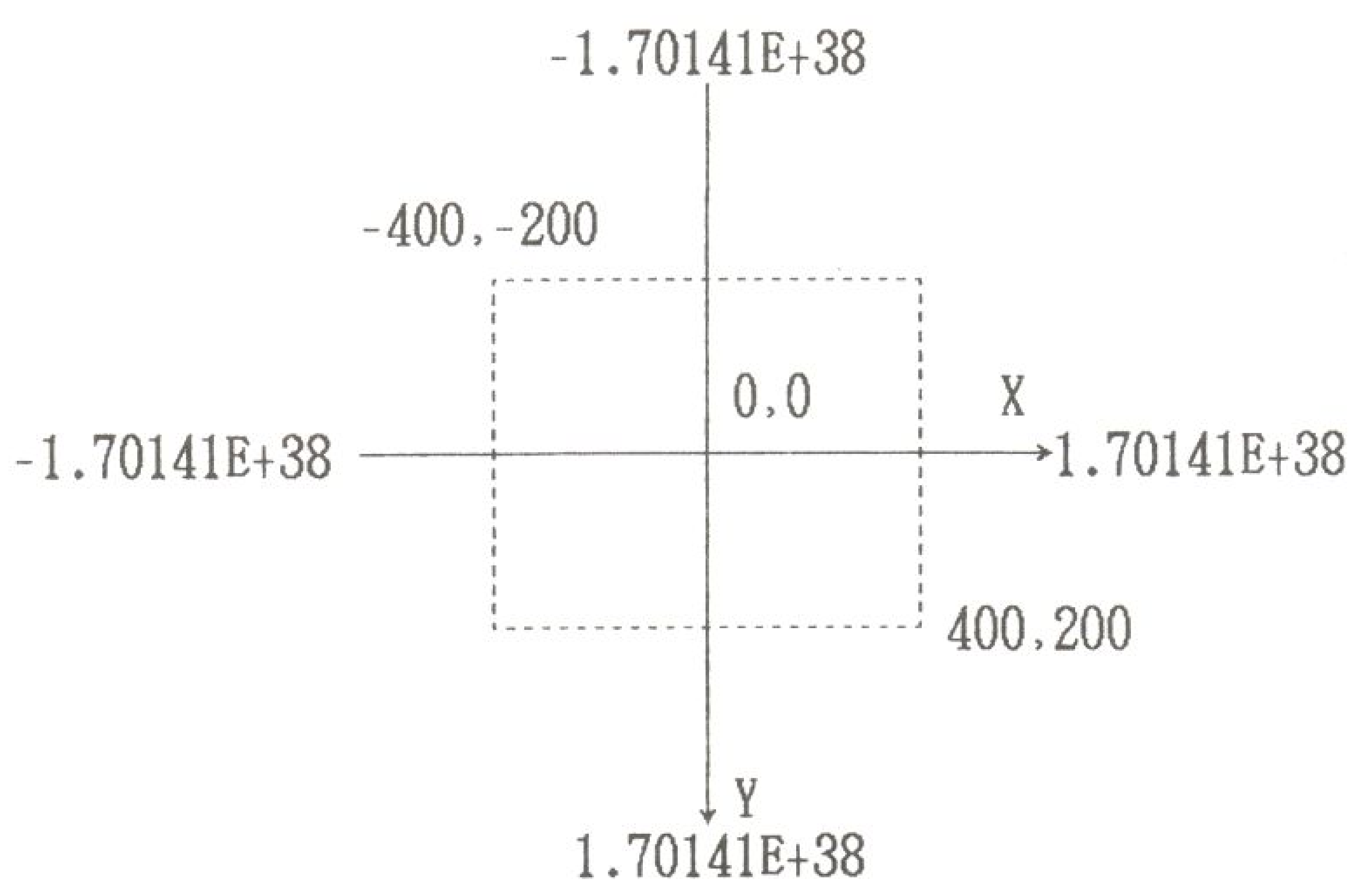
《ワールド座標とウインドー》
　ワールド座標はＸ、Ｙ方向とも単精度の数値の範囲（－１．７０１４１Ｅ＋３８～１．７０１４１Ｅ＋３８）をもつ座標系で、この座標のうちどの部分を表示するかはＷＩＮＤＯＷ文で指定する。ＷＩＮＤＯＷ文で指定した領域を「ウインドー」と呼ぶ。
　ワールド座標を使う命令は、ＷＩＮＤＯＷ、ＰＳＥＴ、ＰＲＥＳＥＴ、ＬＩＮＥ、ＣＩＲＣＬＥ、ＰＡＩＮＴ、ＰＯＩＮＴ。本書ではワールド座標を（Ｗｘ，Ｗｙ）で表す。

〔ＷＩＮＤＯＷ（－４００，－２００）－（４００，２００）と指定した場合〕



《オリジナルスクリーン座標とビューポート》
　グラフィック画面のドットと１対１で対応した座標をオリジナルスクリーン座標と呼ぶ。オリジナルスクリーン座標は（０，０）～（６３９，３９９）の範囲をもつ。
　ワールド座標のウインドーを画面のどの部分に表示するかは、ＶＩＥＷ文で指定する。ＶＩＥＷ文とＶＩＥＷ関数がオリジナルスクリーン座標を使う。

〔ワールド座標とウインドー〕



〔オリジナルスクリーン座標とビューポート〕

ＶＩＥＷ（１００，５０）－（５３９，３４９）と指定した場合

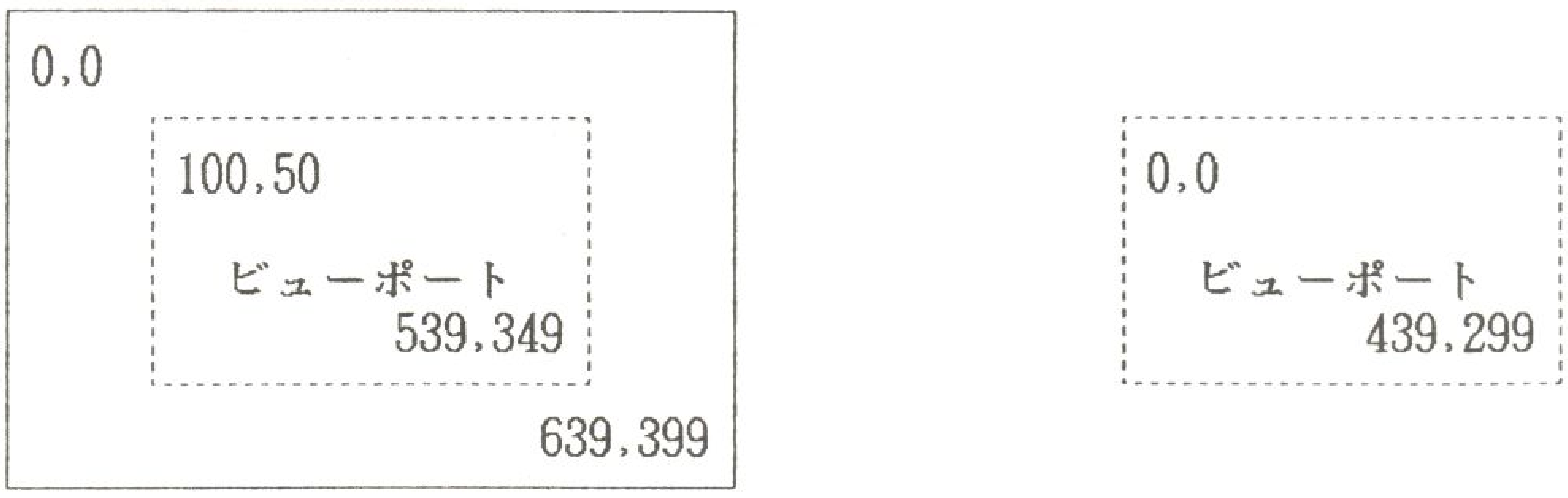


　オリジナルスクリーン座標で指定したディスプレー上の領域（点線内）を「ビューポート」と呼ぶ。ワールド座標で指定したウインドー内にかいた図形は、オリジナルスクリーン座標で指定したビューポート内に表示する。

《スクリーン座標》
　オリジナルスクリーン座標で指定したビューポート内のドットと１対１で対応した座標をスクリーン座標という。スクリーン座標の範囲は、ビューポートの大きさによって違う。スクリーン座標を使う命令は、ＧＥＴ＠、ＰＵＴ＠、ＰＯＩＮＴ関数、ＭＡＰ関数。本書では、スクリーン座標を（Ｓｘ，Ｓｙ）で表す。

〔ＶＩＥＷ（１００，５０）－（５３９，３４９）と指定した場合のスクリーン座標〕

オリジナルスクリーン座標　　　　　　　　スクリーン座標



《座標の指定方法》
　座標の指定方法には、絶対座標と相対座標がある。

・絶対座標
　固定した原点をもとに座標を決める方法を、絶対座標による指定という。キャラクター座標・オリジナルスクリーン座標は絶対座標を使う。ワールド座標・スクリーン座標では、絶対座標のほかに相対座標でも指定できる。
・相対座標
　移動可能な基点を元にして、その基点からのＸ、Ｙ方向の離れ具合で座標の指定ができる。これが相対座標による表し方である。
　相対座標の基点はグラフィック命令で設定される。ＳＴＥＰ（Ｘ，Ｙ）で基点からの相対座標を指定する。ＳＴＥＰ（Ｘ，Ｙ）を一定にして、基点を動かして図形を描くこともできる。
　ＰＳＥＴ、ＰＲＥＳＥＴ、ＬＩＮＥ、ＰＡＩＮＴ、ＣＩＲＣＬＥ、ＰＯＩＮＴ文ではワールド座標の基点を使い、ＧＥＴ＠文はスクリーン座標の基点を使う。

《Ｌａｓｔ　ｒｅｆｅｒｅｎｃｅｄ　Ｐｏｉｎｔ（ＬＰ）》
　ＰＳＥＴ、ＰＲＥＳＥＴ、ＣＩＲＣＬＥ、ＬＩＮＥ、ＰＯＩＮＴ、ＧＥＴ＠、ＰＵＴ＠文を実行すると、ワールド座標とスクリーン座標の基点が同時に変わる。相対座標の基点の位置をＬａｓｔ　ｒｅｆｅｒｅｎｃｅｄ　Ｐｏｉｎｔ（最終参照座標。略してＬＰ）という。

《カラーモード》
　画面に色を出す方法は、テキスト画面とグラフィック画面で違う。

・テキスト画面
　ＣＯＮＳＯＬＥ文でカラーモードにし、ＣＯＬＯＲ文で色を指定する。
・グラフィック画面
　ＳＣＲＥＥＮ文でカラーモードにし、ＰＳＥＴなどの各命令で色を指定する。

《ディスプレー画面の３つの部分》
　ＣＯＬＯＲ文でディスプレー画面の３つの部分の色を指定する。フォアグラウンドカラー、バックグラウンドカラー、ボーダーカラーの３つの色である。

・フォアグラウンドカラー
　グラフィック画面に図形を描くときに使う色。グラフィック命令で色指定のないときは、この色を使う。いわば黒板に図形をかくチョークの色にあたる。
・バックグラウンドカラー
　グラフィック画面の地の色。色指定なしでＰＲＥＳＥＴ文で、点や線を消すと、地のバックグラウンドカラーの色が出てくる。黒板の色と考えるとわかりやすい。
・ボーダーカラー
　ＢＡＳＩＣで扱えない画面の場所の色。黒板がかかっている壁の色のようなもの。

ディスプレー画面



点線内が表示領域

表示領域の色がバックグラウンドカラー

表示領域に描く図形の色がフォアグラウンドカラー

《グラフィック画面のパレットとカラーコード》

グラフィック命令の色の指定にはパレット番号を使う。パレット番号は０～７まであり、ＣＯＬＯＲ文で各パレットとカラーコードを対応づける。ＣＯＬＯＲ文で指定しなければ、パレット番号と同じカラーコードが割り振られている。

各ドットにパレット番号をつけるために３つのプレーンを使う。ドットのパレット番号は、この３つのプレーンをあわせた３ビットで表現する。この３ビットの１０進数値がパレット番号である。

パレット番号とカラーコードの対応を変えると、すでに画面に表示してあった色も自動的に変わる。

グラフィック命令の、ＣＩＲＣＬＥ、ＬＩＮＥ、ＰＡＩＮＴ、ＰＲＥＳＥＴ、ＰＳＥＴ、ＶＩＥＷ文ではパレット番号で色指定する。直接カラーコードで指定するのは、ＣＯＬＯＲ文のテキスト画面の文字の色の指定やパレット番号にカラーコードを割り振るときと、ボーダーカラーの色指定だけである。

色を出すために、赤（Ｒｅｄ）と緑（Ｇｒｅｅｎ）と青（Ｂｌｕｅ）の３つの色を合成して、８色を表現する。

プレーン2のビット	プレーン1のビット	プレーン0のビット	パレット番号
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

ＣＯＬＯＲ文で
パレット番号と
色を対応づける

←------→

緑	赤	青	カラーコード	色
×	×	×	0	黒
×	×	○	1	青
×	○	×	2	赤
×	○	○	3	紫
○	×	×	4	緑
○	×	○	5	水色
○	○	×	6	黄色
○	○	○	7	白

○はその色を表示する
×はその色を表示しない

《デバイス名・入出力命令対応表》

○ ---- 使える
× ---- 使えない
△ ---- 何もしない

	デバイス名	LPT1 :	SCRN :	COM1～3 :	CAS1～2 :	1～10 :	KYBD :
	機器名	プリンター	スクリーン	RS-232C ポート 1～3	カセットテープ	ディスク 1～10	キーボード
入出力命令	BLOAD	×	×	○	×	○	○
	BSAVE	○	○	○	×	○	×
	CLOSE	○	△	○	△	○	△
	EOF関数	×	×	○	×	○	×
	FPOS関数	○	×	×	×	○	×
	GET	×	×	×	○	○	○
	INPUT#	×	×	○	○	○	○
	INPUT $関数	×	×	○	×	○	○
	LINE INPUT#	×	×	○	○	○	○
	LOAD	×	×	○	○	○	○
	LOC関数	×	×	○	×	○	○
	LOF関数	×	×	○	×	○	×
	OPEN	○	△	○	△	○	△
	PRINT#	○	○	○	○	○	×
	PRINT # USING	○	○	○	○	○	×
	PUT	○	○	×	○	○	×
	SAVE	○	○	○	○	○	×
	WIDTH	○	×	○	×	×	×
	WRITE#	○	○	○	○	○	×

（右ページの表の見方の例）
　ＬＩＮＥ文についてみる場合は、右上のＬＩＮＥから下りている点線をたどる。○印の左側が関連する項目となる。
　ＬＩＮＥ文は線を引く命令で、ワールド座標を使う。（Wx1,Wy1 ）－（Wx2,Wy2 ）と指定できるし、第一パラメーターを省略して、－（Wx2,Wy2 ）だけでもよい。（Wx1,Wy1 ）、（Wx2,Wy2 ）について、それぞれSTEP（x1,y1 ）、STEP（x2,y2 ）と相対座標指定できる。また、（Wx1,Wy1 ）－STEP（x2,y2 ）とワールド座標と相対座標を混ぜてもよい。
　実行後のLP（最終参照座標）は（Wx2,Wy2 ）となる。カラー指定にはパレット番号を使い、ラインスタイルを指定できる。

《画面・グラフィック命令と機能・座標・カラー・画面対応表》

・○……画面・グラフィック命令と関連する項目

	CIRCLE	CLS	COLOR	COLOR=	COLOR@	CONSOLE	CSRLIN関数	GET@	LINE	LOCATE	MAP関数	PAINT(1)	PAINT(2)	PEN関数	POINT	POINT(1)関数	POINT(2)関数	POS関数	PRESET	PSET	PUT@	ROLL	SCREEN	VIEW	VIEW関数	WINDOW	WINDOW関数
＜機能＞																											
グラフィック画面のモード設定																							○				
座標変換											○																
LP（最終参照座標）の位置を得る																○											
ウィンドーの設定																							○			○	
ビューポートの設定																							○	○			
ウィンドーの位置																											○
ビューポートの位置																									○		
スクロール																						○					
線を引く	○								○																		
点を打つ																			○	○	○						
色をぬる		○										○	○									○		○			
スクリーン上のパレット番号を得る																	○										
パレット番号とカラーコードの対応をとる				○																							
＜ワールド座標＞																											
ワールド座標を使う	○								○		○	○	○		○	○			○	○						○	○
(Wx,Wy)	○											○	○		○				○	○							
(Wx1,Wy1) — (Wx2,Wy2)									○																	○	
— (Wx2,Wy2)									○																		
(Wx,Wy) →STEP (x,y)	○											○	○		○				○	○							
(Wx1,Wy1) →STEP (x1,y1)									○																		
(Wx2,Wy2) →STEP (x2,y2)									○																		
＜オリジナルスクリーン座標＞																											
オリジナルスクリーン座標を使う																						○		○	○		
(Sx1,Sy1) — (Sx2,Sy2)																								○			
＜スクリーン座標＞																											
スクリーン座標を使う								○			○					○	○				○						
(Sx,Sy)																	○				○						
(Sx1,Sy1) — (Sx2,Sy2)								○																			
(Sx2,Sy2) →STEP (x2,y2)								○																			
＜実行後のLP（最終参照座標）の位置＞																											
(Wx,Wy)	○											○	○		○				○	○							
(Wx2,Wy2)									○																		
(Sx,Sy)																					○						
(Sx2,Sy2)								○																			
ビューポートまたはウィンドーの左上点		○																					○	○		○	
＜カラー＞																											
パレット番号を使う	○		○	○					○			○	○				○		○	○	○			○			
カラーコードを使う			○	○	○																						
バックグラウンドカラー		○	○																○			○					
フォアグラウンドカラー	○		○									○	○							○							
ボーダーカラー（カラーコード指定）			○																								
ラインスタイル									○																		
タイルストリング													○														
＜テキスト画面・キャラクター座標＞																											
(X1 ,Y1) — (X2 ,Y2)					○																						
スクロールウィンドー設定						○																					
スクロールウィンドーをクリアー		○																									
文字の色・カラーコード指定			○																								
四角形内の色・ファンクションコード指定					○																						
カーソルの水平位置										○				○				○									
カーソルの行位置							○			○				○													

２．文字と演算子

	文　　字	意　　味
機能	？　（疑問符）	ＰＲＩＮＴと同じ。Ｌ？は（９８）ならＬＰＲＩＮＴとなる
	’　（アポストロフィ）	ＲＥＭと同じ。行の途中からでも使える ＰＲＩＮＴ　２＋５　’タシザ゛ン
	．　（ピリオド）	現在の行番号。たとえばプログラムがエラー終了したとき、 ＬＩＳＴ　．でエラーとなった行を表示する
	：　（コロン）	１行に複数の文（マルチステートメント）を入れるときの区切り記号 １０　Ａ＝１０：Ｂ＝２０
	；　（セミコロン）	ＰＲＩＮＴ文での区切り記号 １０　ＰＲＩＮＴ”コタエ＝”；Ａ
	，　（コンマ）	命令文にパラメーターがあるときの区切り記号 ＷＩＤＴＨ　８０，２５ ＰＲＩＮＴ文での区切り記号。１４文字ずつ区切る １０　ＰＲＩＮＴ　Ａ，Ｂ
	－　（ハイフン）	行の範囲を指定する ＬＩＳＴ　１００－５００
	＊　（アスタリスク）	ラベル名の第１桁目につける １０　ＧＯＴＯ　＊ＬＡＢ ３０　＊ＬＡＢ
型宣言	＆Ｏ　又は　＆	８進定数を宣言する。ＯはＯｃｔａｌの略 ＆はアンパーサントと読む
	＆Ｈ	１６進定数を宣言する。ＨはＨｅｘａ　Ｄｅｃｉｍａｌの略 ＆Ｈ５１など
	”　（引用符）	文字定数を宣言する ”ストリング”　０～２５５文字の範囲 ＰＲＩＮＴ　”ＡＳＡＨＩ”
	Ｅ	単精度実数型定数を宣言する ？１．５Ｅ＋５は１．５×１０＾５のことで１５００００
	Ｄ	倍精度実数型定数を宣言する ？－１．５Ｄ－５は－．００００１５
	＄	ストリング（文字列）型変数を宣言する Ａ＄など　ストリングは０～２５５文字の範囲
	％	整数型変数、定数を宣言する Ａ％や１２３％など　－３２７６８～３２７６７の範囲
	！	単精度実数型変数、定数を宣言する Ａ！や１２３！など　有効桁数７桁の浮動小数値
	＃	倍精度実数型変数、定数を宣言する Ａ＃や１．２３＃など　有効桁数１６桁の浮動小数値

	文　　字	意　　味
算術演算	＝	右辺の結果を左辺へ代入する　A＝B＋10など
	－	負数表示または減算　－1．5やA＝B－10など
	＋	加算または文字列の結合　A＝B＋10 B＄＝”XY”：A＄＝B＄＋”AB”など
	＊	乗算　C＝A＊BやA＝A＊BやA＝23＊65など
	／	結果が浮動小数点の除算　B＝A／10など
	＾	べキ乗　B＝A＾2　は　B＝A＊Aと同じ
	￥	結果が整数の除算　C＝A￥B　（11￥3は3）
	MOD	結果が整数の剰余　C＝A　MOD　B （11　MOD　3は2）
論理演算	NOT	否定、全ビット反転。結果は整数 B＝NOT　A NOT　2は－3
	AND	論理積。結果は整数 C＝A　AND　B（11　AND　6は2）
	OR	論理和。結果は整数 C＝A　OR　B（11　OR　6は15）
	XOR	排他的論理和。どちらか一方にビットが立っていれば1、両方とも0、または両方とも1は0。結果は整数 C＝A　XOR　B（11　XOR　6は13）
	EQV	等価。XORの否定形。結果は整数 C＝A　EQV　B（11　EQV　6は－14）
	IMP（インプリケーション）	X　IMP　Y　はYが1なら1、Yが0のときはXの否定 Z＝X　IMP　Y（5　IMP　4は－2）結果は整数
関係演算	＝　＞　＜ ＜＞　＞＜ ＝＜　＞＝ ＝＞　＜＝	条件判定　整数　真なら－1、偽なら0 A＝B　（5＝3は0） A＝＞B（7＝＞4は－1） A＞＜B（6＞＜6は0）

真理値表

NOT論理
(B = NOT A)

A	B
0	1
1	0

AND論理
(C = A AND B)

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

OR論理
(C = A OR B)

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

XOR論理
(C = A XOR B)

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

EQV論理
(C = A EQV B)

A	B	C
0	0	1
0	1	0
1	0	0
1	1	1

IMP論理
(C = A IMP B)

A	B	C
0	0	1
0	1	1
1	0	0
1	1	1

3. BASICのコマンドとステートメント

AUTO　〔＜開始行＞〔，〔＜間隔＞〕〕〕

行番号を自動発生させる。停止は【CTRL】＋【C】または【STOP】。

AUTO ………………… 10、20、30、40 …… と発生する。
AUTO　50，50 … 50、100、150 …… と発生する。
AUTO　100 ……… 100、110、120 …… と発生する。
AUTO　200，…… 200から直前で実行した間隔で発生する。

BEEP〔0または1〕

内蔵ブザーから音を出す。

BEEP　1 … 連続して鳴る。BEEP　0で停止する。BEEPだけなら約0．4秒鳴る。

BLOAD＜”デバイス名：ファイル名”＞〔，＜オフセット＞〕〔，R〕

（ディスク）

機械語プログラムをデバイスからメモリーへロードする。

オフセット…ロードを開始する番地のオフセットを指定する。省略すると、BSAVEで指定したオフセット。開始番地はDEF　SEG文で指定したセグメントベース番地に、オフセットを加えた番地になる。次項参照。
（88）ロードを開始する番地を指定する。省略すると、BSAVEで指定した番地。
R ………… それまで開いていたファイルはそのままで、実行する。省略するとロードだけする。

BLOAD”1：PRG”（ディスク番号1番は省略可。BLOAD”PRG”でも同じ）

BSAVE＜”デバイス名：ファイル名”＞，＜オフセット＞，＜長さ＞

（ディスク）

機械語プログラムをメモリーからデバイスへセーブする。ファイル名は使用者が各自でつけたもの。

オフセット…セーブする機械語プログラムの開始番地のオフセットを指定する。
オフセットは、DEF　SEG文で指定したセグメントベース番地からの変位。
（88）セーブする機械語プログラムの開始番地を指定する。開始番地は、機械語プログラムが入っている番地。
長さ ……… セーブする機械語プログラムのバイト数を指定する。



アドレス（98）　10　DEF　SEG＝&H1000
1E410H　　20　BSAVE”1：PRG”，&HE400，&H11
セグメントベース番地は実際の番地を16で割った値で指定する。上例では&H10000ではなく、&H1000。DEF　SEG文参照。（88）にはセグメントの考え方はない。
1E400H
10000H　　（88）10　BSAVE”1：PRG”，&HE000，&H11

CALL　＜変数名＞〔（＜引数名＞〔，＜引数名＞・・・・〕）〕

（ディスク）

BASICのプログラムの中から機械語サブルーチンを呼ぶ。（98）あらかじめ、DEF　SEG文でセグメントベース番地を定義する。次にそのセグメントの中の機械語サブルーチン開始番地のオフセットを変数に入れる。実際のサブルーチン開始番地は、セグメントベース番地＋オフセット。（88）機械語サブルーチンの開始番地を指定する。変数名に入っている値が開始番地。
「USR関数とCALL文の使い方」参照。

変数名 … 機械語サブルーチンの開始番地が入っている変数。配列変数は使えない。
引数名 … 機械語サブルーチンへ引き渡す変数名を指定する。定数や式は引数として使えない。

```
10　CLEAR　，&H1F00
20　DEF　SEG＝&H1000
30　BSUB＝&HF000：B＝100：C＝200
40　CALL　BSUB（B，C）
```

```
CHAIN 〔MERGE〕<”デバイス名:ファイル名”>
        〔〔,<行番号>〕〔,ALL〕〔,DELETE<範囲>〕〕
```
(ディスク)

プログラムをロードし、実行する。すでに開かれたファイルはそのまま。CHAIN文を使えば、いくつかに分けたプログラムを、次つぎに実行できる。ALLもMERGEも指定しないと、DEFINT文などの宣言文は無効になる。

- 行番号 …… 実行を開始する行番号。省略するとプログラムの最初から実行する。
- オプション ALL …… ロードしたプログラムにすべての変数、配列を渡す。省略すると、なにも引き渡さない。COMMON文参照のこと。
- MERGE …… プログラムをロードし、メモリー上のプログラムと混ぜ合わせ、行番号から実行する。行番号を省略すると、混ぜ合わせたプログラムの最初から実行する。プログラムは、アスキー形式でSAVEしたもの。
- DELETE …… MERGEを指定したとき有効。ディスクのプログラムをロードする前に、メモリー上のプログラムの指定した範囲を削除する。
- 範囲 …… 行番号(例100-230など)、またはラベル名(*ABC-*XYZ)で指定する。

```
10 A$=”**ABC**”:B=123
20 CHAIN”1:PRG1”,ALL
20 CHAIN”1:PRG1”,,ALL (88)
```

ディスク1番の中にPRG1の名前を持つプログラムがある。そのプログラムは次のとおり

```
10 PRINT A$;B
```

上のプログラムを実行した結果
ABC 123
メモリー上のプログラムは、10 PRINT A$;B に入れ替わる。

```
10 CHAIN MERGE”2:PRG2”,30,DELETE 40-50
30 PRINT ”CC”;
40 PRINT ”DD”;
50 PRINT ”EE”;
```

ディスク2番の中にPRG2の名前を持つプログラムがある。そのプログラムは次のとおり

```
20 PRINT ”AA”;
30 PRINT ”BB”;
```

上のプログラムを実行すると、20行は追加、30行はPRG2の30行とおきかわる。40行50行は削除。メモリー上のプログラムは、次のようになる。

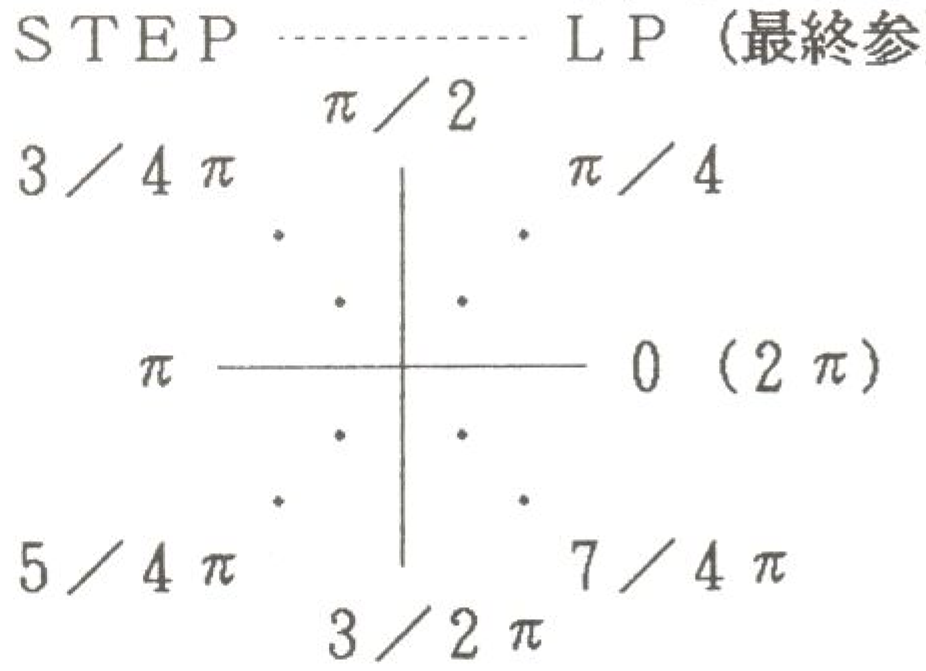
```
10 CHAIN MERGE”2:PRG2”,30,DELETE 40-50
20 PRINT ”AA”;
30 PRINT ”BB”;
```

実行結果は BB

```
CIRCLE (Wx,Wy) またはSTEP (X,Y) ,<半径>〔,<パレット番号>〕
        〔,<開始角>〕〔,<終了角>〕〔,<比率>〕
```

ワールド座標Wx,Wyを中心に指定半径の円、円弧、扇形をパレットの色で描く。

- 半径 …… WINDOW文で指定した水平方向の長さの単位で半径の大きさが決まる。
- パレット番号 …… 省略するとフォアグラウンドカラー。
- 開始角,終了角 …… 弧または扇形を描く。角はラジアンで-2π～2π。省略すると0,2πで円。0,πは半円弧。負値で指定すると扇形。
- 比率 …… 垂直方向半径/水平方向半径の値。半径の長さは画面上のドットで決まる。真円は640×200画面では比率0.5。640×400画面で1。指定しないとそれぞれ0.5と1。楕円にするには比率を変える。
- STEP …… LP(最終参照座標。POINT文参照)から、中心がX,Yだけ移動する。



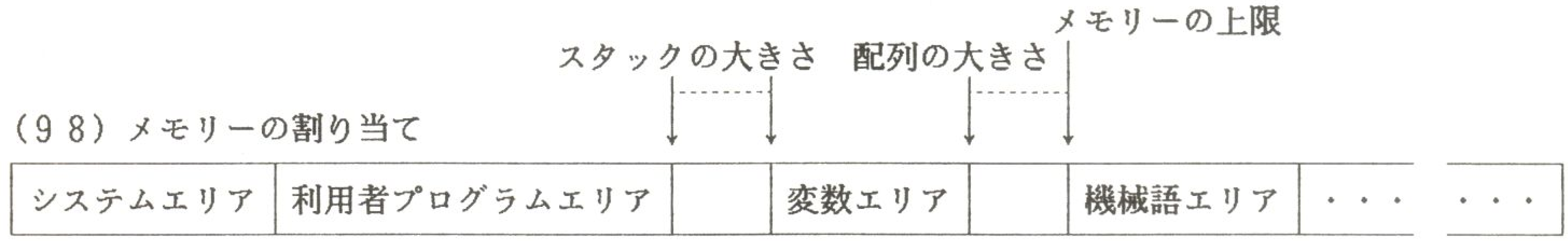
扇形と縦長の楕円と円を描く

```
10 WINDOW(0,0)-(639,399)
20 CLS 2
30 CIRCLE(200,200),90,,-3.141
   6/4,-3/4*3.1416
40 CIRCLE(200,200),90,,,,2
50 CIRCLE STEP(30,30),50
```

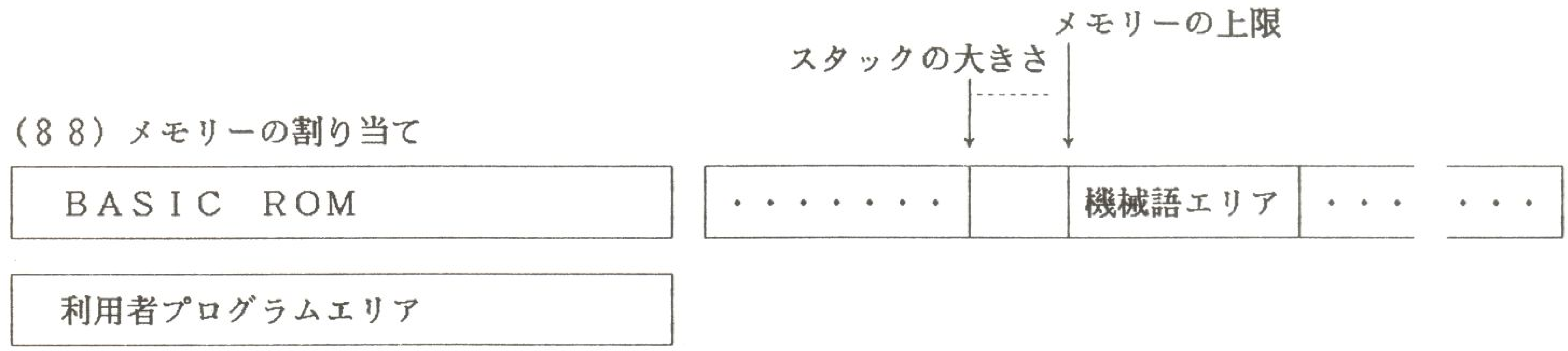
（９８）CLEAR〔，〔＜メモリーの上限＞〕〔，＜スタックの大きさ＞〕
〔，＜配列の大きさ＞〕〕
（８８）CLEAR〔，〔＜メモリーの上限＞〕〔，＜スタックの大きさ＞〕〕

数値変数を０、文字変数をヌルストリング（””）にクリアーして、ＢＡＳＩＣプログラムのメモリー領域の上限、作業用スタック、配列の大きさを定義する。機械語を乗せる場合にメモリーの上限を移動する。ＤＩＭ文で大きな配列定義になる場合、プログラム実行中ＦＯＲやＧＯＳＵＢで作業スタックが不足してエラーになった場合などに使う。「メモリーアドレスマップ」参照。

メモリーの上限 ……… セグメントベース番地で指定。ＰＣ－８００１と違いメモリーの上限は、ＢＡＳＩＣ側に含まれない。
（８８）指定した番地がメモリーの上限。メモリーの上限はＢＡＳＩＣ側に含まれない。
スタックの大きさ …… バイト数で指定。
配列の大きさ ………… バイト数／１６で大きさを指定。



CLEAR　，&HE０００（メモリーの上限をＥ００００Ｈ番地にした。ここから機械語が入る）



CLEAR　，&HE５００（メモリーの上限をＥ５００Ｈ番地にした。ここから機械語が入る）

CLOSE　〔〔＃〕＜ファイル番号＞〔，〔＃〕＜ファイル番号＞・・・・〕〕

ファイルへの入出力を終了する。ファイル番号をすべて省略すると、オープン状態のすべてのファイルを閉じる。ＯＰＥＮと対で使う。ＥＮＤ文、ＮＥＷ命令でもクローズする。ＳＴＯＰ文は一時停止だからファイルを閉じない。

CLOSE　＃１，＃２

CLS　〔＜機能＞〕

画面を消す。機能を省略すると１。
機能　１ …… テキスト画面のスクロールウインドー（せり上がり部分）をクリアーする。
　　　２ …… グラフィック画面のビューポートをバックグラウンドカラーでクリアーする。ＶＩＥＷ文参照。
　　　３ …… テキスト・グラフィック両画面をクリアーする。

COLOR　〔＜ファンクションコード＞〕〔，＜バックグラウンドカラー＞〕
〔，＜ボーダーカラー＞〕〔，＜フォアグラウンドカラー＞〕

画面の色または機能を指定する。
ファンクションコード ……… テキスト画面の文字の色を０～７で指定する。白黒モードとカラーモードで機能がちがう。モードの指定はＣＯＮＳＯＬＥ文でしておく。

コード	白黒モード	カラーモード
0	ノーマル	黒
1	シークレット（文字は表示しない）	青
2	ブリンク（点滅する）	赤
3	シークレット（1と同じ）	紫
4	リバース（反転する）	緑
5	リバースシークレット（反転して文字は表示しない）	水色
6	リバースブリンク（反転して点滅）	黄色
7	リバースシークレット（5と同じ）	白

バックグラウンドカラー……グラフィック画面の地の色をパレット番号で指定する。ＣＬＳ文で画面をクリアーすると、地の色はここで指定した色になる。
ボーダーカラー……画面周囲の枠の色をカラーコードで指定する。ただし、高解像度ディスプレーを使用している場合は、0（黒）のみ指定できる。通常指定しない。
フォアグラウンドカラー……グラフィック画面に点や線を描くときに使う色を、パレット番号で指定。

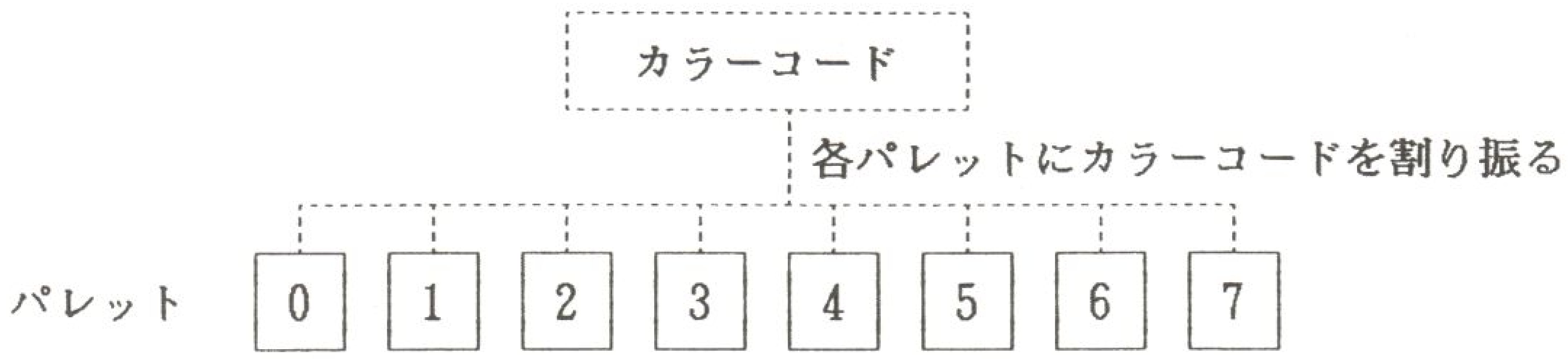
ＣＯＬＯＲ＝（<パレット番号>，<カラーコード>）

8個あるパレット（絵の具入れ）にカラーコード（色）をセットする。グラフィック画面で使い、いちど描いた図形の色を置き換える。電源投入時はパレット番号＝カラーコード。

カラーコード

0……黒	4……緑		
1……青	5……水色		
2……赤	6……黄色		
3……紫	7……白		

```
10　SCREEN　0
20　COLOR＝（1，4）
30　LINE（100，100）－（200，250），1
40　FOR　J＝1　TO　200：NEXT　J
50　COLOR＝（1，2）　これで緑の線が赤になる。
```



ＣＯＬＯＲ＠（Ｘ1，Ｙ1）－（Ｘ2，Ｙ2）〔，<ファンクションコード>〕

テキスト画面の2点を対角とする四角形内の文字に色（カラーモード）または機能（白黒モード）をつける。ファンクションコードは、ＣＯＬＯＲ文のファンクションコードと同じ。

ＣＯＭ　ＯＮまたはＯＦＦまたはＳＴＯＰ

ＲＳ－２３２Ｃコミュニケーションポートによる割り込みを許可・禁止または一時停止する。ＯＮ　ＣＯＭ　ＧＯＳＵＢ文と対で使う。
ＯＮ……割り込みを許可する。
ＯＦＦ……割り込みを禁止する。
ＳＴＯＰ……割り込みを一時停止する。一時停止の状態で発生した割り込みは、ＣＯＭ　ＯＮ文を実行すれば有効になる。

ＣＯＭＭＯＮ　<変数>〔，<変数>・・・〕　（ディスク）

ＣＨＡＩＮ文で実行するプログラムに引き渡したい変数を定義する。配列変数は”（　）”をつける。ＣＨＡＩＮ文の前で宣言する。

```
10　DIM　D（20）：A＝34：B＝78：C$＝”XYZ”
20　COMMON　A，B，C$，D（）
30　CHAIN　”1：SUB1”
```

ＣＯＮＳＯＬＥ　〔＜スクロール開始行＞〕，〔＜スクロール行数＞〕 〔，＜ファンクションキー表示＞〕〔，＜カラー白黒スイッチ＞〕

テキスト画面のモード、スクロールウインドーの大きさなどを設定する。スクロールウインドーとは、画面をせり上げたりせり下げたりできる領域のこと。ＣＬＳ　１やＰＲＩＮＴ　ＣＨＲ＄（１２）はスクロールウインドー内をクリアーする。

スクロール開始行 ……………… スクロールを開始する行を指定する。
スクロール行数 ………………… スクロールをさせる行数を指定する。
ファンクションキー表示 ……… 0　画面下のファンクションキー表示を消す。
　　　　　　　　　　　　　　 1　ファンクションキーを表示する。
カラー白黒スイッチ …………… 0　白黒モード。
　　　　　　　　　　　　　　 1　カラーモード。

ＣＯＮＴ

ＳＴＯＰキー、ＳＴＯＰ文または【ＣＴＲＬ】＋【Ｃ】で停止したプログラムの実行を再開する。停止中にプログラムを変更すると、ＣＯＮＴ命令による再開はできない。

ＣＯＰＹ　〔＜機能＞〕

画面の文字や図形をプリンターへ出力する。機能を省略すると３。キー配置図のＣＯＰＹキー参照。

機能　1 …… テキスト画面をプリンターへ出力する。
　　　2 …… グラフィック画面をプリンターへ出力する。
　　　3 …… テキスト画面とグラフィック画面をプリンターへ出力する。
　　　4 …… 640 ×200 ドットのとき、上下を½に縮小したグラフィック画面をプリンターへ出力する。漢字は読みやすい。640 ×400 ドットのときはかわらない。
　　　5 …… テキスト画面とグラフィック画面をプリンターへ出力する。（漢字出力用でグラフィック画面については機能４と同じ）

ＤＡＴＡ　＜定数＞〔，＜定数＞・・・〕

ＲＥＡＤ文で読まれる数値や文字を割り当てる。ＲＥＡＤ文と対で使う。

```
１０　ＲＥＡＤ　Ａ，Ｂ＄，Ｃ
２０　ＰＲＩＮＴ　Ａ；Ｂ＄；Ｃ
３０　ＤＡＴＡ　１０，”ＡＢＣ”，２０　　”ＡＢＣ”はＡＢＣでも可。
```

ＤＥＦ　ＦＮ＜関数名＞〔（＜引数の並び＞）〕＝＜関数の定義式＞

ユーザー関数を定義する。

```
１０　ＤＥＦ　ＦＮＥＮＭＥＮＳＥＫＩ（Ｒ）＝３．１４＊Ｒ＾２
２０　Ｘ＝ＦＮＥＮＭＥＮＳＥＫＩ（５）
３０　ＰＲＩＮＴ　Ｘ
```

関数名をＥＮＭＥＮＳＥＫＩとした。変数Ｒは１０行のなかだけで対応していればよい。

（９８）ＤＥＦ　ＳＥＧ〔＝＜セグメントベース番地＞〕

セグメントのベース番地を指定する。メモリーの番地を指定するときは、セグメントベース番地とオフセット（セグメントベース番地からの変位）を組み合せる。実際の番地（物理番地）は、セグメントベース番地×１６＋オフセットになる。ＰＥＥＫ，ＰＯＫＥ，ＤＥＦ　ＵＳＲ，ＣＡＬＬ文などでは、セグメントベース番地に対するオフセットを指定するので、これらの命令の前に指定する必要がある。

たとえば、ＤＥＦ　ＳＥＧ＝＆Ｈ１Ｆ８１：ＰＯＫＥ　＆Ｈ１Ａ２，１とすると１を書き込む実際の番地は

＆Ｈ１Ｆ８１×１６＋＆Ｈ１Ａ２＝＆Ｈ１Ｆ８１０＋＆Ｈ１Ａ２
　　　　　　　　　　　　　　　＝＆Ｈ１Ｆ９Ｂ２

になる。

```
ＤＥＦ　ＵＳＲ〔＜数字＞〕＝＜番地＞
```

機械語サブルーチンの開始番地を定義する。ＣＬＥＡＲ文で定義したメモリーの上限から使える。
　数字 ………… ０から９まで。省略すると０。
　番地 ………… オフセットを指定する。実際の開始番地は、ＤＥＦ　ＳＥＧ文で指定したセグメントベース番地にこの値を足した番地。
　（８８）直接の番地。

```
10  DEF  USR2=&HE000
 :
80  Y=USR2 (X)
```

ＢＡＳＩＣでＸを入力すると、機械語サブルーチンが計算して、結果をＹに戻す。ＢＡＳＩＣとサブルーチンの仲立ちはレジスターがする。「ＵＳＲ関数とＣＡＬＬ文の使い方」参照。

```
ＤＥＦＩＮＴ　＜文字＞〔－＜文字＞〕〔，＜文字＞〔－＜文字＞〕・・・・〕
ＤＥＦＳＮＧ　＜文字＞〔－＜文字＞〕〔，＜文字＞〔－＜文字＞〕・・・・〕
ＤＥＦＤＢＬ　＜文字＞〔－＜文字＞〕〔，＜文字＞〔－＜文字＞〕・・・・〕
ＤＥＦＳＴＲ　＜文字＞〔－＜文字＞〕〔，＜文字＞〔－＜文字＞〕・・・・〕
```

指定した文字で始まる変数の型を上から整数型、単精度実数型、倍精度実数型、文字型に定義する。ただし、型宣言文字（％、！、＃、＄）がつねに定義に優先する。
　ＤＥＦＩＮＴ　Ａ－Ｅ　　　　　Ａ、Ｂ、Ｃ、Ｄ、Ｅの各文字で始まる変数を整数型に定義する。

```
ＤＥＬＥＴＥ　〔＜開始行＞〕〔－＜終了行＞〕
```

プログラムの指定行を抹消する。開始行を省略したときはプログラムの最初の行から終了行まで。終了行を省略したときは開始行だけ抹消する。

　ＤＥＬＥＴＥ　１００－２００ ……… １００行から２００行を抹消する。
　ＤＥＬＥＴＥ　－３００ ……………… ３００行までを抹消する。
　ＤＥＬＥＴＥ　４００ ………………… ４００行を抹消する。４００【ＲＥＴ】と同じ。

```
ＤＩＭ　＜変数名＞（＜要素の数＞〔，＜要素の数＞・・・〕）
　　　　　　　　〔，＜変数名＞（＜要素の数＞〔，＜要素の数＞・・・〕）・・・〕
```

配列の大きさを決めて、メモリーに領域を確保する。配列の添字の下限は通常０。ＯＰＴＩＯＮ　ＢＡＳＥ文で下限を０か１に変更できる。

　ＯＰＴＩＯＮ　ＢＡＳＥ　０
　ＤＩＭ　Ａ（３）…………

Ａ（０）	Ａ（１）	Ａ（２）	Ａ（３）

まで４個確保。

　ＯＰＴＩＯＮ　ＢＡＳＥ　１
　ＤＩＭ　Ｙ（２，３）………… ２次元で確保。

Ｙ（１，１）	Ｙ（１，２）	Ｙ（１，３）
Ｙ（２，１）	Ｙ（２，２）	Ｙ（２，３）

```
ＤＳＫＯ＄　＜ドライブ番号＞〔，＜サーフェス番号＞〕，＜トラック番号＞
　　　　　　　　　　　　　　　　　　　　　　　　　，＜セクター番号＞
```
（ディスク）

ドライブ、サーフェス（両面フロッピーディスクのときだけ指定する。０は表、１は裏）、トラック、セクターを指定し、そのセクターにＦＩＥＬＤ＃０で指定したランダムバッファの内容を書きこむ。ファイル・アロケーション・テーブルやディレクトリーと無関係に動くので危険。
　ＤＳＫＯ＄はステートメントだが、逆の機能を持つディスク関数にＤＳＫＩ＄がある。

ＥＤＩＴ　<行番号または＊ラベル名>または．

　指定した行を画面表示し、カーソルを表示した行に移す。ＤＩＳＫ－ＢＡＳＩＣのときは、Roll up、Roll downキーで指定した行の前後をスクロールさせて表示することができる。

```
ＥＤＩＴ　１００ …… １００行を表示する。
ＥＤＩＴ　＊ＬＡＢ … ＊ＬＡＢの行を表示する。
ＥＤＩＴ　．………… エラーになったときなどの該当行を表示する。
```

ＥＮＤ

　プログラムの実行を終了し、ファイルをすべてクローズする。ＥＮＤ文は省略できるが、ファイルはクローズされない。

ＥＲＡＳＥ　<配列名>〔，<配列名>・・・・〕

　配列を抹消する。したがって、そのメモリースペースを別用途に使える。

```
１０　ＤＩＭ　Ａ（３）
２０　Ａ（１）＝１０：ＰＲＩＮＴ　Ａ（１）
３０　ＥＲＡＳＥ　Ａ：ＤＩＭ　Ａ（２，３）　　配列Ａを抹消して、配列Ａを再定義した
```

ＥＲＲＯＲ　〔<エラーコード>〕

　エラー発生をシミュレートする。ＯＮ　ＥＲＲＯＲ　ＧＯＴＯ文と組みにして使う。ＢＡＳＩＣで使っていないエラーコードを定義して、エラー処理ルーチンで使う。ＢＡＳＩＣで使っているエラーコードと重複しないように注意する。「ＢＡＳＩＣのエラーメッセージ」参照。

```
１０　ＯＮ　ＥＲＲＯＲ　ＧＯ　ＴＯ　＊ＥＲＳＵＢ
２０　ＩＮＰＵＴ　Ａ
３０　ＩＦ　Ａ＞１００　ＴＨＥＮ　ＥＲＲＯＲ　２００……… エラーを発生させる。
４０　ＰＲＩＮＴ　Ａ：ＧＯＴＯ　２０
５０　＊ＥＲＳＵＢ　’エラーショリ
６０　ＩＦ　ＥＲＲ＝２００　ＴＨＥＮ　ＢＥＥＰ ……… ここで使う。
７０　ＲＥＳＵＭＥ　２０
```

ＦＩＥＬＤ〔＃〕<ファイル番号>，<フィールドの大きさ>　ＡＳ　<文字変数>
〔，<フィールドの大きさ>　ＡＳ　<文字変数>・・〕

ランダムファイルバッファの中に変数用領域を割り当てる。

ファイル番号 ………… ＯＰＥＮ文で指定したファイル番号または０。
フィールドの大きさ … 文字変数に割り当てるバイト数。
　整数型のとき ………… ２バイト（ＭＫＩ＄・ＣＶＩ参照）
　単精度実数型のとき … ４バイト（ＭＫＳ＄・ＣＶＳ参照）
　倍精度実数型のとき … ８バイト（ＭＫＤ＄・ＣＶＤ参照）
　文字型のとき ………… 任意。
文字変数 ……………… 割り当てる文字変数名。

```
ＦＩＥＬＤ　＃１，１０　ＡＳ　Ａ＄，２０　ＡＳ　Ｂ＄，４　ＡＳ　Ｃ＄
```

　ランダムバッファの最初の１０バイトをＡ＄に、次の２０バイトをＢ＄に、次の４バイトをＣ＄に割り当てる。合計で２５６バイトまで割り当てられる。
　ＦＩＥＬＤ文の変数に値を代入するときは、ＬＳＥＴかＲＳＥＴ文を使う。ＩＮＰＵＴ文やＬＥＴ文で代入すると文字領域と混ざりあって、ランダムバッファの位置を正しく指さなくなる。

```
FILES 〔<ドライブ番号>〕
```
（ディスク）

指定したドライブ番号のディスクに存在するファイル名、種類、大きさ（クラスターの数）を表示する。ドライブ番号を省略すると１番。ファイルの種類はファイル名の末尾（ただし拡張子の前）に表示。

種類………… 空白　アスキー形式でセーブしたプログラムまたはデータファイル。
　　　　　　．　　バイナリー形式でセーブしたプログラムファイル。
　　　　　　＊　　ＢＳＡＶＥ命令でセーブした機械語プログラムファイル。

ドライブ１のＦＩＬＥＳをとる

```
FILES 1
      PROG2*ASM   1   ←…BSAVE命令でセーブした機械語プログラムファイル
```

```
FOR <変数名>=<式> TO <式>〔STEP <式>〕
```

ＮＥＸＴ文までの処理を条件によって繰り返す。ＳＴＥＰ＜式＞は、増加（減少）の幅。なければ１ずつ増加する。ＦＯＲ～ＮＥＸＴと対で使う。

```
10 FOR K=0 TO 4 STEP 2
20 PRINT K;
30 NEXT K
```

左のプログラムを実行すると、
０　２　４　と結果が表示される。

```
FOR K=・・・  ──┐
FOR L=・・・  ─┐│
               ││
NEXT L     ───┘│
NEXT K     ─────┘
```

多重ループの場合、内側のＮＥＸＴ文は、外側のＮＥＸＴ文の前にあること。

```
(1) FOR K=1  TO 1  STEP 0
(2) FOR K=10 TO 1  STEP 1
(3) FOR K=1  TO 10 STEP -5
```

これらの場合、ＦＯＲ～ＮＥＸＴの文は実行しないで、ＮＥＸＴ文の次の文に実行が移る。

```
GET〔#〕<ファイル番号>〔,<式>〕
```

ファイルからデータをバッファに読み込む。ファイルがディスクとそれ以外では、機能が異なる。ディスクファイルかディスクファイルでないかは、ＯＰＥＮ文で指定したデバイス名で決まる。

ファイル番号………………………… ＯＰＥＮ文で指定したファイル番号を指定する。

ディスクファイルのとき………… ランダムファイルから１レコード読み込み、ランダムバッファに入れる。同時にＦＩＥＬＤ文で指定した各フィールドに対応した変数に、そのデータを代入する。式はレコード番号。式を省略すると最後にＧＥＴ、ＰＵＴした次のレコード番号のレコードが、バッファに読み込まれる。

ディスクファイルでないとき…… カセットテープなどのデバイスからデータをバッファに読み込む。式は読み込む文字数。式を０と指定するか、省略すると２５６になる。

```
10 OPEN "1:DATA" AS #1
20 FIELD#1,6 AS A$,2 AS B$
30 GET #1,2
40 PRINT A$;CVI(B$)
50 CLOSE #1
```

```
GET〔@〕(Sx1,Sy1)-(Sx2,Sy2)またはSTEP(X,Y),<配列変数名>
                                        〔(<要素>)〕
```

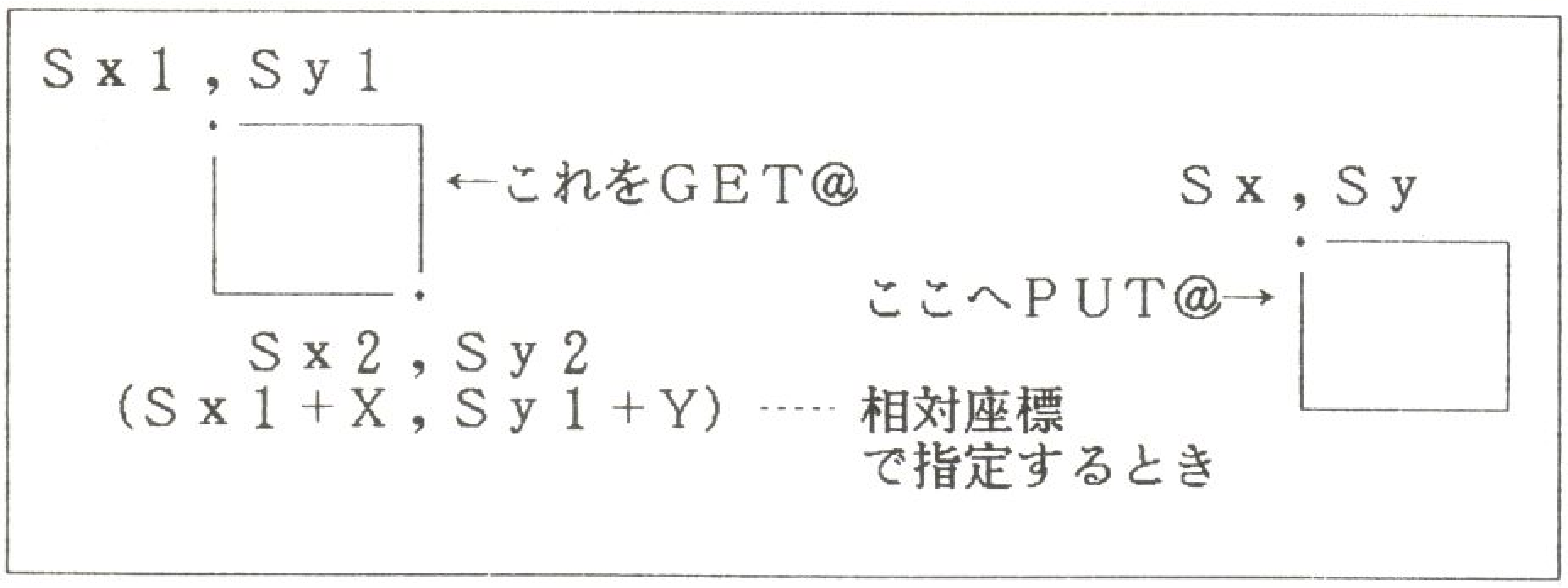
スクリーン座標の２点を対角とする四角形内のグラフィックパターンを配列変数に読み込む。配列変数はあらかじめ最低限必要な大きさを確保しておくこと。

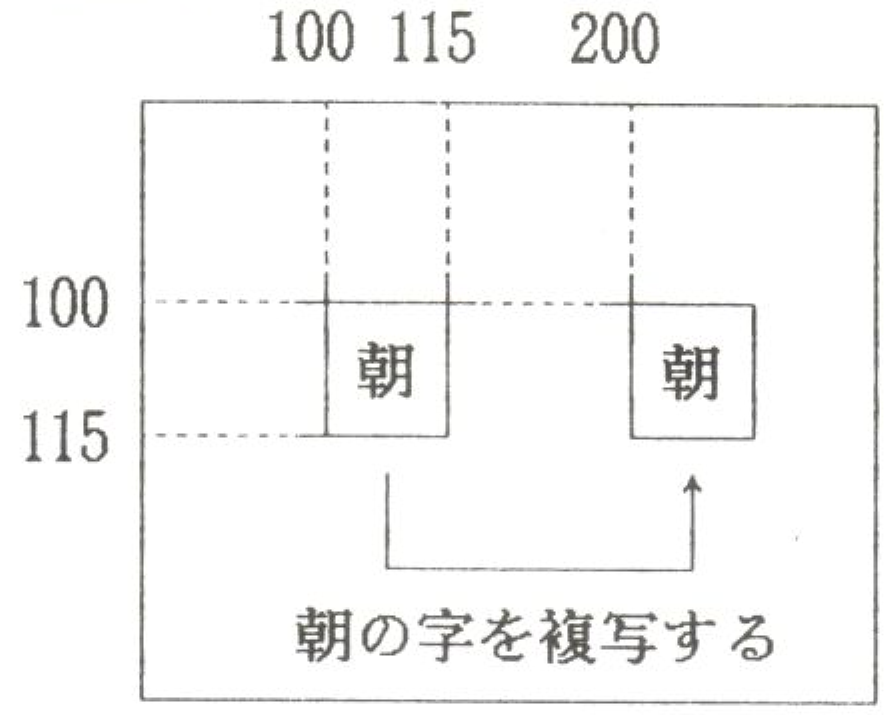
配列変数名………………… グラフィックパターンを読み込む配列変数名を指定する。

要素…………………………… 配列の途中から読み込むとき指定。省略すると配列の最初から読み込む。

ＳＴＥＰ…………………… （Ｓｘ１，Ｓｙ１）からの相対座標を指定する。

ＰＵＴ＠やＬＩＮＥ文などで描いたグラフィックパターンを配列に読み込む。

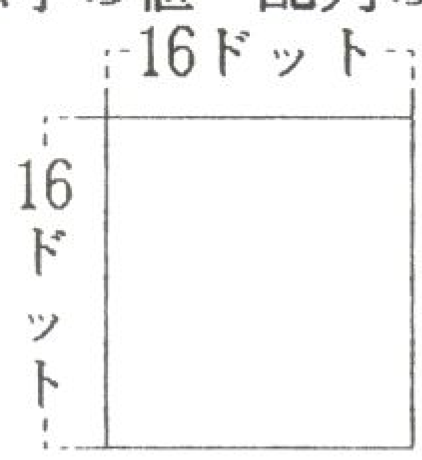




配列の大きさの計算のしかた（ＯＰＴＩＯＮ　ＢＡＳＥ　０のとき）

必要なバイト数＝（（横ドット数＋７）￥８）×縦ドット数×Ｍ＋４
配列の大きさ　＝必要なバイト数￥Ｌ＋１
添字の値（ＤＩＭ文のカッコの中の数）＝配列の大きさ－１
（ＯＰＴＩＯＮ　ＢＡＳＥ　１のときは添字の値＝配列の大きさ）

Ｍ …… 白黒モードのとき …………… １
　　　 カラーモードのとき …………… ３
Ｌ …… 整数型配列のとき …………… ２
　　　 単精度実数型配列のとき …… ４
　　　 倍精度実数型配列のとき …… ８



上例の朝の漢字（16ドット×16ドット）を整数型配列Ａ％（Ｉ）でとるには
（（１６＋７）￥８）×１６×３＋４＝１００
１００￥２＋１＝５１
したがってＩ＝５１－１、Ａ％（５０）

```
10 SCREEN 0,0:CLS 2
20 DIM A%(50)
30 PUT@(100,100),KANJI(&H442B)     …… "朝"を描く
40 GET@(100,100)-(115,115),A%      …… "朝"を読み込む
50 PUT@(200,100),A%                …… "朝"を複写
```

ＧＯＳＵＢ　＜行番号または＊ラベル名＞

指定した行番号から始まるサブルーチンへ実行を移す。サブルーチンの出口には必ずＲＥＴＵＲＮ文が必要。このＲＥＴＵＲＮ文でＧＯＳＵＢ文の次の文に実行が戻る。

ＧＯＴＯ　＜行番号または＊ラベル名＞

指定した行番号へプログラムの制御を移す。

ＨＥＬＰ　ＯＮまたはＯＦＦまたはＳＴＯＰ

ＨＥＬＰキーによる割り込みを許可・禁止または一時停止する。ＯＮ　ＨＥＬＰ　ＧＯＳＵＢ文と対で使う。

ＯＮ ……… 割り込みを許可する。
ＯＦＦ …… 割り込みを禁止する。
ＳＴＯＰ … 割り込みを一時停止する。一時停止の状態で発生した割り込みは、ＨＥＬＰ　ＯＮ文を実行すれば有効になる。

```
10 HELP ON
20 ON HELP GOSUB 100
30 INPUT A:PRINT A*A
40 PRINT A+A
50 GOTO 30
 :
100 PRINT "WARIKOMI*HELP":RETURN
```

３０行のＩＮＰＵＴ文実行中にＨＥＬＰキーを押すと、１００行を実行したあと
（９８）はＰＲＩＮＴ　Ａ＊Ａ
（８８）はＰＲＩＮＴ　Ａ＋Ａから実行する。

ＩＦ　＜条件式＞〔ＴＨＥＮ〕＜文または行番号または＊ラベル名＞
〔ＥＬＳＥ＜文または行番号または＊ラベル名＞〕

条件式が真ならＴＨＥＮのあとの命令、偽ならＥＬＳＥのあとの命令を実行する。
ＴＨＥＮの後がＧＯＴＯ文のときのみＴＨＥＮを省略できる。

①ＩＦ　Ａ＞０　ＴＨＥＮ　『文１』	Ａ＞０のときだけ『文１』を実行。
②ＩＦ　Ａ＞０　ＴＨＥＮ　『文１』　ＥＬＳＥ　『文２』	Ａ＞０のとき『文１』を、Ａ＜＝０のとき『文２』を実行。
③ＩＦ　Ａ＝０　ＴＨＥＮ　『文１』 ＥＬＳＥ　ＩＦ　Ｂ＞０　ＴＨＥＮ　『文２』 ＥＬＳＥ　『文３』	Ａ＝０のとき『文１』を、Ａ＜＞０のときは、ＥＬＳＥの後のＩＦ文を実行。
④ＩＦ　Ａ　ＴＨＥＮ　『文１』　ＥＬＳＥ　『文２』	Ａが真（０以外）のとき『文１』を偽（０）のとき『文２』を実行。

ＩＮＰＵＴ〔＜”プロンプト文”＞，または；〕＜変数＞〔，＜変数＞・・・・〕

キーボードからデータを入力する。プロンプト文と変数をセミコロン（；）で区切るとプロンプト文につづき、疑問符（？）と１個の空白を表示して入力待ちになる。コンマ（，）で区切るとプロンプト文だけ表示して入力待ちになる。データを入れずにリターンキーを押すと、変数の型に対応して数値変数は０、文字変数はヌルストリング（””）が入る。
変数を複数指定したときは、データをコンマ（，）で区切って入力する。

```
１０　ＩＮＰＵＴ　”データ　ヲ　ドウゾ”；Ａ，Ｂ＄
２０　ＰＲＩＮＴ　Ａ，Ｂ＄
```

ＩＮＰＵＴ　ＷＡＩＴ＜時間＞，〔＜”プロンプト文”＞，〕＜変数＞〔，＜変数＞・・・・〕

キーボードからのデータ入力に時間制限をつける。時間内に入力がなかったときは次の行から実行する。時間は、０．１秒単位で指定する。マルチステートメントにすると、後の文は指定した時間内に入力があったときだけ実行する。

```
１０　ＩＮＰＵＴ　ＷＡＩＴ　１００，Ａ＄：ＰＲＩＮＴ　Ａ＄：ＥＮＤ
２０　ＰＲＩＮＴ　”＊＊＊”
```

上のプログラムを実行し、１０秒以内に入力しないと　＊＊＊　を表示する。

ＩＮＰＵＴ　＃＜ファイル番号＞，＜変数＞〔，＜変数＞・・・・〕

シーケンシャルデータファイルから変数にデータを読み込む。ファイル番号は、ファイルを開いた（ＯＰＥＮ）ときに指定したファイル番号。

ＫＥＹ　＜キー番号＞，＜”文字列”＞

ｆ・１からｆ・１０のファンクションキーを定義する。最大１５文字まで文字列とコントロール文字を指定できる。

```
ＫＥＹ　１，”ＷＩＤＴＨ　４０，２０”＋ＣＨＲ＄（１３）
ＫＥＹ　２，”ＲＥＭ＊＊＊＊＊”
```

ＫＥＹ　ＬＩＳＴ

ファンクションキーのｆ・１からｆ・１０の内容を表示する。

KEY 〔(<キー番号>)〕ONまたはOFFまたはSTOP

ファンクションキーによる割り込みを許可・禁止または一時停止する。ON KEY GOSUB文と対で使う。

キー番号 …… ファンクションキーの番号、1～10を指定する。
省略すると、すべてのファンクションキーが対象となる。
ON ………… 割り込みを許可する。
OFF ……… 割り込みを禁止する。
STOP …… 割り込みを一時停止する。一時停止の状態で発生した割り込みは、KEY ON文を実行すれば有効になる。

```
  10 KEY (1) ON:KEY (2) ON
  20 ON KEY GOSUB 100,200
   :
  90 GO TO 20
 100 PRINT "**F1**"
 110 RETURN
 200 PRINT "**F2**"
 210 RETURN
```

ファンクションキー1を押すと100行、2を押すと200行に実行が移る。

KEY ONの状態でプログラムが終わると、ファンクションキーが使えない。ダイレクトモードで、KEY OFFとすればよい。

KILL<"ドライブ番号:ファイル名"> (ディスク)

指定したドライブ番号の指定したファイルを削除する。ドライブ番号1番は省略可。ファイルは、プログラムファイルでもデータファイルでもよい。クローズされたファイルしか削除できない。OPEN状態だと"File already open"エラーが起きる。

```
KILL"2:PROG5"
```

(98) KINPUT <変数>

キーボードから日本語データを入力する。入力待ちの状態でキーボードからJISコード(1文字当たり4桁)を入力すると、対応した日本語文字を1文字ずつ表示する。JISコード4桁を入力したあとで、その文字を修正したいときは、カーソルを2桁戻し正しいコード4桁を入力すればよい。ただし、JISコード4桁を入れる途中(たとえば3桁目)でカーソル移動キーを押すと、その1文字は入力エラーになり、ブザーがなって再入力待ちになる。再度入力すればよい。入力の終わりはリターンキーを押す。(88)にはこの機能はない。

```
10 KINPUT K$
20 PRINT K$
```

〔LET〕<変数名>=<式>

変数に式の値を代入する。LETは省略できる。

```
10 LET A=5*9
20 B=7*8
30 PRINT A;B
```

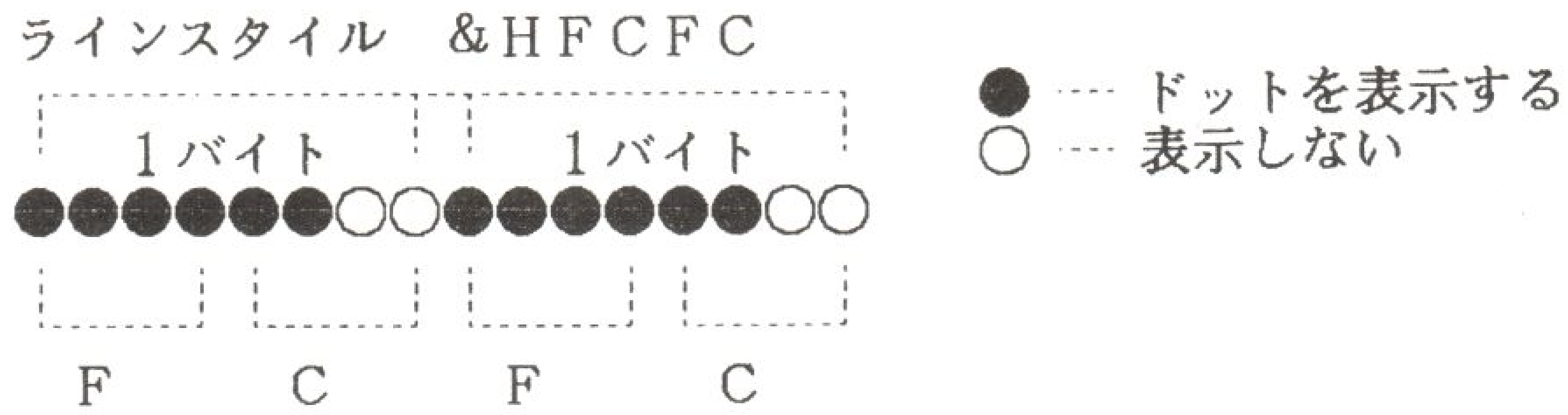
LFILES 〔<ドライブ番号>〕 (ディスク)

FILES命令と同じ内容をプリンターに出力する。

ＬＩＮＥ〔（Ｗｘ１，Ｗｙ１）またはＳＴＥＰ（ｘ１，ｙ１）〕－（Ｗｘ２，Ｗｙ２）または ＳＴＥＰ（ｘ２，ｙ２）〔，＜パレット番号＞〕〔，Ｂまたは，ＢＦ〕〔，＜ラインスタイル＞〕

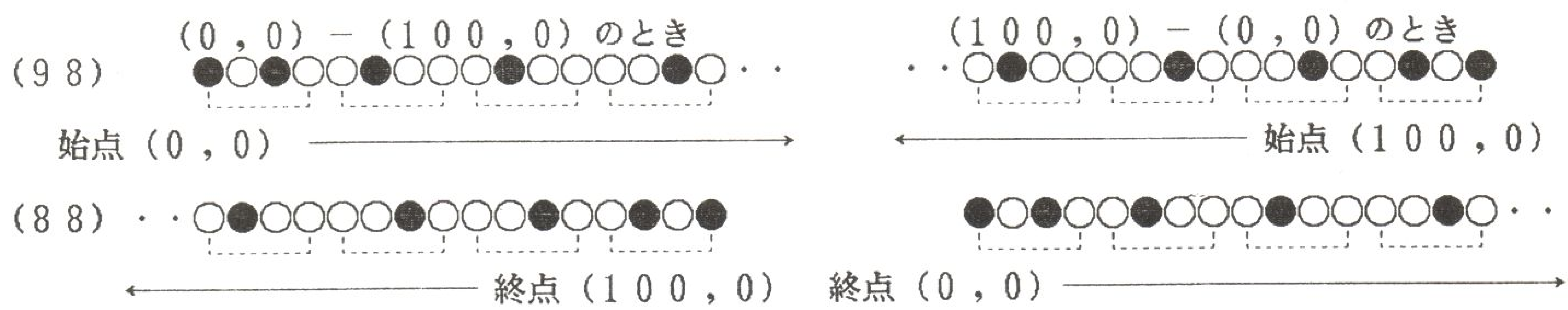
ワールド座標の２点を指定して線や箱を描く。（Ｗｘ１，Ｗｙ１）またはＳＴＥＰ（ｘ１，ｙ１）を省略すると最終参照座標（ＬＰ）から描く。

- パレット番号……省略するとフォアグラウンドカラー。
- Ｂ……２つの座標を対角とする四角形を描く。
- ＢＦ……２つの座標を対角とする四角形を描き中を塗りつぶす。
- ラインスタイル……２バイトの整数で指定。＆Ｈ０～＆ＨＦＦＦＦの範囲。点線や鎖線を引ける。ラインスタイルを指定したときは、ＢＦの指定はできない。省略すると直線になる。指定した値のビットパターンが画面のドットに対応する。



ＳＴＥＰ……最終参照座標（ＬＰ）から（ｘ１，ｙ１）、（ｘ２，ｙ２）を加えた座標。

同じラインスタイルを指定しても（９８）と（８８）で違ったパターンになる。たとえば＆ＨＡ４４２を指定すると次のようになる。




```
10  SCREEN  0,0:CLS  2
20  LINE(10,10)-(600,180),1,B
30  LINE(320,50)-(160,100),2
40  LINE  -(320,150),3
50  LINE  -(480,100),4,,&H3F20
60  LINE  -(320,50),5,,&H20F0
70  LINE  STEP(-40,30)-STEP(80,40),6,BF
```

ＬＩＮＥ　ＩＮＰＵＴ〔＜”プロンプト文”＞，または　；〕＜文字変数＞

キーボードから特殊文字を含めて１行全体（２５５文字以内）を入力する。

ＬＩＮＥ　ＩＮＰＵＴ　ＷＡＩＴ＜時間＞，〔＜”プロンプト文”＞，または；〕＜文字変数＞

ＬＩＮＥ　ＩＮＰＵＴに時間制限をつける。時間の指定および、マルチステートメントに関する制約は、ＩＮＰＵＴ　ＷＡＩＴ文と同じ。

```
10  LINE  INPUT  WAIT  100,A$      時間制限は10秒
```

ＬＩＮＥ　ＩＮＰＵＴ＃＜ファイル番号＞，＜文字変数＞

シーケンシャルデータファイルから１行全体（２５５文字以内）を文字変数に読み込む。ファイル番号はＯＰＥＮ文で指定したファイル番号。

```
LIST 〔<開始行>〕〔-〔<終了行>〕〕
```

開始行から終了行までのプログラムを表示する。

LIST ………………………………… 全部表示。
LIST . ……………………………… エラーになったときなどの該当行を表示。
LIST .- …………………………… 該当行からを表示。
LIST -. …………………………… 該当行までを表示。
LIST 300 ………………………… 300行を表示。
LIST 300- ……………………… 300行からを表示。
LIST -300 ……………………… 300行までを表示。
LIST 100-300 ………………… 100行から300行を表示。
LIST *LAB1-*LAB2 ……… *LAB1行から*LAB2行を表示。

```
LLIST 〔<開始行>〕〔-〔<終了行>〕〕
```

プリンターへプログラムを出力する。使い方はLIST命令と同じ。

```
LOAD<"デバイス名:ファイル名">〔,R〕
```

プログラムをデバイス名で指定した装置からメモリーへロードする。

デバイス名…… CAS1またはCAS …… 転送速度1200ボーのカセットテープ。
　　　　　　　CAS2 ……………………… 転送速度600ボーのカセットテープ。
　　　　　　　COM1～3 ……………… RS-232Cポート。
　　　　　　　1～10 ……………………… フロッピーデイスク。値はドライブ番号。
　　　　　　　　　　　　　　　　　　　（88）フロッピーディスクは1～8。
R ……………… それまで開いていたファイルはそのままで、プログラムをロードし実行する。省略するとロードだけ。

LOAD"CAS:PRG" ……PRGという名前のプログラムをカセットテープからロードする。
LOAD"1:PRG1" ………ディスクのドライブ1に入っているフロッピーから、PRG1というプログラムをロードする。

```
LOAD?<"デバイス名:ファイル名">
```

プログラムをカセットテープにSAVE（記録）した後、正しくSAVEされたかどうか検証する。メモリーの内容は消えない。ファイル名はSAVEのとき指定したプログラムの名前。

デバイス名 …… CAS1またはCAS …… 転送速度1200ボーのカセットテープ。
　　　　　　　CAS2 ……………………… 転送速度600ボーのカセットテープ。

LOAD?"CAS:PRG"

```
LOCATE〔<水平位置>〕〔,<垂直位置>〕〔,<カーソルスイッチ>〕
```

指定したキャラクター座標にカーソルを移す。キャラクター座標は、画面の左上を（0,0）とする。

水平位置 ……………… 省略すると0。
垂直位置 ……………… 省略すると現在の行。
カーソルスイッチ …… カーソルを表示するとき1、表示しないとき0。省略すると前の状態。

```
10 LOCATE 30,10
20 PRINT "ABC"
```

```
LPRINT〔<式>〕〔,または;<式>・・・・〕
```

文字や数値をプリンターへ出力する。PRINT文参照。
（98） L?でも同じ。

ＬＰＲＩＮＴ　ＵＳＩＮＧ＜”フォーマット文”＞；＜式＞〔　，または　；　＜式＞・・・・〕

文字や数値を編集してプリンターへ出力する。「ＰＲＩＮＴ　ＵＳＩＮＧ一覧表参照」

ＬＳＥＴ　＜文字変数＞＝＜文字列＞

ＦＩＥＬＤ文で割り当てたランダムバッファの文字変数へ文字列を左づめで移す。数値データは移す前にＭＫＩ＄・ＭＫＳ＄・ＭＫＤ＄のいずれかを使って文字型にしなくてはならない。文字変数は、あらかじめＦＩＥＬＤ文で割り当てておく。

```
40 FIELD #1,10 AS A$,6 AS B$,2 AS C$,4 AS D$
50 LSET A$="COFFEE"
60 LSET B$=X$
70 LSET C$=MKI$(N%)
80 LSET D$=MKS$(2046)
```

ＭＥＲＧＥ＜”デバイス名：ファイル名”＞　（ディスク）

メモリー上のプログラムに、デバイスのプログラムファイルを混ぜ合わせる。同じ行番号は、デバイスのプログラムと置き変わる。デバイスのプログラムは、アスキー形式でＳＡＶＥしたもの。

メモリー上のプログラム	ディスクのプログラム
10 INPUT A	30 C=B*30
20 B=A*10	40 PRINT C
30 PRINT B	50 END
40 END	

この２つのプログラムをＭＥＲＧＥすると、以下のようになる。

```
10 INPUT A
20 B=A*10
30 C=B*30
40 PRINT C
50 END
```

ＭＯＮ

ＢＡＳＩＣモードから機械語モニターモードへ切りかえる。
ＢＡＳＩＣモードに戻るには【ＣＴＲＬ】＋【Ｂ】。
モニターモードで使えるコマンドについては、「モニターの使い方」参照。

ＭＯＴＯＲ　〔＜スイッチ＞〕

カセットテープレコーダーのモーターを制御する。スイッチを省略するとｏｎのときｏｆｆに、ｏｆｆのときｏｎにする。

スイッチ	値	動作
	０	モーターがとまる（ｏｆｆ）
	０以外の数	モーターが動く（ｏｎ）

ＮＡＭＥ　＜”デバイス名：旧ファイル名”＞ＡＳ＜”デバイス名：新ファイル名”＞　（ディスク）

ディスク上のファイル名を変更する。デバイス名を省略するとドライブ１番。

ＮＥＷ

メモリー上のプログラムと変数をクリアーする。

NEW　ON　<式>

システム立ち上げ時にディップスイッチで設定した機能を、ソフトウェアで変更する。たとえばN88－BASICからN－BASICを起動するときは、NEW　ON　1。

（98）　式の値は8ビットでSW2を表し、1ビットがディップスイッチ1つに対応する。（ビットが1のときスイッチオン、0はスイッチオフ）

（88）　16ビットでSW1とSW2を表す。

詳細は、「準備作業」参照。

NEXT〔<ループ変数名>〔，<ループ変数名>・・・　〕〕

FORループの範囲を決める。FOR文参照。

ON　COM　GOSUB　<行番号または＊ラベル名>〔，<行番号または＊ラベル名>〕〔，<行番号または＊ラベル名>〕

RS－232Cコミュニケーションポートに入力があったときに実行する割り込み処理ルーチンの開始行を指定する。指定した順番がコミュニケーションポート1、2、3にそれぞれ対応する。この命令が有効なのはCOM　ONであらかじめ割り込みを許可してある場合。

ON　ERROR　GOTO　<行番号または＊ラベル名>または0

エラーが発生したときに実行するエラー処理ルーチンを指定する。エラー処理ルーチンの出口はRESUME文。ON　ERROR　GOTO　0を実行するとそれ以降エラーが発生しても、エラー処理ルーチンは実行しない。エラーメッセージを表示してプログラムは終了する。ただし、エラー処理ルーチンの中でON　ERROR　GOTO　0とすると、その時点でエラーメッセージを表示してプログラムは終了する。

```
 10 ON ERROR GOTO 100
 20 INPUT A,B          Aの値が0のとき30行でエラーが起こる。
 30 PRINT B/A          （エラーコード＝11）
 40 END
100 IF ERR=11 THEN A=1   Aに1を入れてエラーを起こした行へ戻る
110 RESUME

105 ON ERROR GOTO 0   追加するとエラーメッセージを表示してプログラムは終了
```

ON　<式>　GOSUB　<行番号または＊ラベル名>〔，<行番号または＊ラベル名>・・・・〕

式の値に対応したサブルーチンを実行する。式の値が0または指定した行番号の数より大きいときには次の行から実行する。ただし、式の値が負のときにはエラーになる。

```
ON A-1 GOSUB 100,200,500
```

A－1の値が1なら100行、2なら200行、3なら500行のサブルーチンを実行し次の行へ戻る。負のときにはエラー、その他の値だと次の行へ移る。

ON　<式>　GOTO　<行番号または＊ラベル名>〔，<行番号または＊ラベル名>・・・・〕

式の値に対応した行番号から実行する。ON　GOSUB文参照。

ON　HELP　GOSUB　<行番号または＊ラベル名>

HELPキーが押されたとき実行する割り込み処理ルーチンの開始行を指定する。この命令が有効なのはあらかじめHELP　ON文で割り込みを許可してある場合。HELP　ON文参照。

```
ON KEY GOSUB <行番号または*ラベル名>〔,<行番号または*ラベル名>・・・・〕
```

ファンクションキーが押されたとき実行する割り込み処理ルーチンの開始行を指定する。指定した行番号の順番とファンクションキーの番号が対応する。この命令が有効なのはあらかじめＫＥＹ　ＯＮ文で割り込みを許可してある場合。ＫＥＹ　ＯＮ文参照。

```
ON PEN GOSUB <行番号または*ラベル名>
```

ライトペンが押されたとき実行する割り込み処理ルーチンの開始行を指定する。この命令が有効なのはあらかじめＰＥＮ　ＯＮ文で割り込みを許可してある場合。ＰＥＮ　ＯＮ文参照。

```
ON STOP GOSUB <行番号または*ラベル名>
```

ＳＴＯＰキーが押されたとき実行する割り込み処理ルーチンの開始行を指定する。この命令が有効なのはあらかじめＳＴＯＰ　ＯＮ文で割り込みを許可してある場合。ＳＴＯＰ　ＯＮ文参照。

```
ON TIME$=<"hh:mm:ss">GOSUB<行番号または*ラベル名>
```

リアルタイムタイマーによる割り込み発生時刻と、割り込み処理ルーチンの開始行を指定する。この命令が有効なのはＴＩＭＥ＄　ＯＮ文で割り込みが許可されている場合。ＴＩＭＥ＄　ＯＮ文参照。ＯＮ　ＴＩＭＥ＄　ＧＯＳＵＢ文を実行した時点で指定した時刻から内部クロック（ＴＩＭＥ＄）を引いた時刻を計算する。計算した時刻を経過したら割り込み処理ルーチンを実行する。±１秒程度の誤差がある。

```
10 TIME$ ON
20 TIME$="12:00:00"
30 ON TIME$="12:05:00" GOSUB 500
40 TIME$="11:00:00"
```

３０行……５分後に割り込み処理ルーチンを実行。
４０行……内蔵クロックは変わるが、３０行目で計算した割り込み時刻は変わらない。

```
OPEN <"デバイス名:ファイル名"> 〔FOR <モード>〕 AS〔#〕<ファイル番号>
```

ファイルの使用宣言をする。デバイス名を省略するとドライブ１番または、ＣＡＳ１。
- モード　ＩＮＰＵＴ……入力ファイルの指定。ファイルポインターはそのファイルの先頭を指す。ファイルがないとエラーになる。
- ＯＵＴＰＵＴ……新しい出力ファイルを作る。ファイルがあれば古いファイルを削除する。ファイルポインターは先頭を指す。
- ＡＰＰＥＮＤ……追加。ファイルポインターはそのファイルの最後のレコードの直後を指す。ファイルがないとエラーになる。
- 省略すると……ファイルがあればファイルポインターは先頭を指す。ファイルがなければ新しいファイルを作りファイルポインターは先頭を指す。

```
OPTION BASE 0 または 1
```

配列の添字の下限を０か１に設定する。ＯＰＴＩＯＮ　ＢＡＳＥ文がなければ、添字の下限は０。

```
OUT <ポート番号>,<データ>
```

出力ポートに１バイトのデータ（０～２５５）を出力する。

```
(98) OUT &H37,&H92 …… BEEP 1
     OUT &H37,&H9B …… BEEP 0
(88) OUT &H40,&H20 …… BEEP 1
     OUT &H40,&H00 …… BEEP 0
```

ＰＡＩＮＴ　（Ｗｘ，Ｗｙ）またはＳＴＥＰ（Ｘ，Ｙ）〔，＜領域色＞〔，＜境界色＞〕〕

ワールド座標で指定した点を含む境界色で囲まれた領域を、領域色で塗りつぶす。ＰＡＩＮＴ文はビューポート内だけを塗る。指定した座標がウインドーの外にあるとエラー。

　領域色……パレット番号で指定。省略するとフォアグラウンドカラー。
　境界色……パレット番号で指定。省略すると領域色と同じ。
　ＳＴＥＰ…最終参照座標（ＬＰ）からの相対座標。
指定した座標がすでに領域色ならば、塗らずに終わる。

```
１０　ＳＣＲＥＥＮ　０，０：ＣＬＳ　２
２０　ＷＩＮＤＯＷ（０，０）－（６３９，１９９）
３０　ＶＩＥＷ（１００，５０）－（５００，１５０），，１
４０　ＬＩＮＥ（１５０，０）－（４５０，１５０），，Ｂ
５０　ＰＡＩＮＴ（１６０，１０），３，７
６０　ＬＩＮＥ（３００，１００）－（６００，２００），２，Ｂ
７０　ＰＡＩＮＴ（３１０，１１０），４，２
```

（９８）ＰＡＩＮＴ（Ｗｘ，Ｗｙ）またはＳＴＥＰ（Ｘ，Ｙ），＜タイルストリング＞〔，＜境界色＞〕

（８８）ＰＡＩＮＴ（Ｗｘ，Ｗｙ）またはＳＴＥＰ（Ｘ，Ｙ），＜タイルストリング＞〔，＜境界色＞〕〔，＜バックグラウンド＞〕

ワールド座標で指定した点を含む境界色で囲まれた領域にタイルをしきつめる。タイルの大きさと模様はタイルストリングで指定する。

　タイルストリング
　　白黒モードのとき
　　　横８ドットを１バイト（８ビット）で表す。ビットが１のとき、対応するドットは白、ビットが０のとき、対応するドットは黒。縦のドット数分のタイルストリングを指定する。

```
ＣＨＲ＄（＆ＨＣＣ）＋ＣＨＲ＄（＆Ｈ３３）
　＆ＨＣＣ　１１００１１００……●●○○●●○○
　＆Ｈ３３　００１１００１１……○○●●○○●●
```

　　カラーモードのとき（パレット番号＝カラーコードの場合）
　　　横８ドットを３バイトで表す。先頭からそれぞれ青・赤・緑のドットパターンに対応する。縦のドット数は、タイルストリングのバイト数／３。ただし、３バイトに満たないとエラーになり、３バイト以上で余りがあれば、余りの分は無視される。

```
ＣＨＲ＄（＆Ｈ４Ｄ）＋ＣＨＲ＄（＆Ｈ２Ｂ）＋ＣＨＲ＄（＆Ｈ１７）
　青　＆Ｈ４Ｄ　０１００１１０１
　赤　＆Ｈ２Ｂ　００１０１０１１　○○○○○○○○
　緑　＆Ｈ１７　０００１０１１１　黒青赤緑紫水黄白
　　　　　　　　　　　　　　　　　　　　　　色色
```

　境界色…………パレット番号で指定。省略するとフォアグラウンドカラー。
　ＳＴＥＰ………最終参照座標（ＬＰ）からの相対座標。
　バックグラウンド…既にタイリングされている領域を異なるパターンでタイリングするとき、タイリングされているタイルストリングを指定する。

ＰＥＮ　ＯＮまたはＯＦＦまたはＳＴＯＰ

ライトペンによる割り込みを許可・禁止または一時停止する。ＯＮ　ＰＥＮ　ＧＯＳＵＢ文と対で使う。

　ＯＮ……割り込みを許可する。
　ＯＦＦ……割り込みを禁止する。
　ＳＴＯＰ…割り込みを一時停止する。一時停止の状態で発生した割り込みは、ＰＥＮ　ＯＮ文を実行すれば有効になる。

ＰＯＩＮＴ（Ｗｘ，Ｗｙ）またはＳＴＥＰ（Ｘ，Ｙ）

最終参照座標（ＬＰ：Ｌａｓｔ　ｒｅｆｅｒｅｎｃｅｄ　Ｐｏｉｎｔ）を（Ｗｘ，Ｗｙ）に変更する。
　ＳＴＥＰ……最終参照座標（ＬＰ）からの相対座標。

ＰＯＫＥ　＜メモリー番地＞，＜データ＞

メモリー内の指定した番地にデータを書き込む。指定した番地の表示はＰＥＥＫ関数参照。
　メモリー番地……オフセットを指定する。＆Ｈ００００から＆ＨＦＦＦＦの範囲。ＤＥＦ　ＳＥＧ文参照。
　　（８８）メモリーの番地。＆Ｈ００００から＆ＨＦＦＦＦの範囲。
　データ……１バイトのデータで＆Ｈ００から＆ＨＦＦの範囲。

ＰＲＥＳＥＴ（Ｗｘ，Ｗｙ）またはＳＴＥＰ（Ｘ，Ｙ）〔，＜パレット番号＞〕

ワールド座標で指定したドットをパレット番号の色で塗りかえる。ＰＳＥＴ文参照。
　パレット番号……省略したときは、バックグラウンドカラーの色で塗りかえる。
　ＳＴＥＰ……最終参照座標（ＬＰ）からの相対座標。

```
10 SCREEN 0,0:CLS 2
20 FOR Y=0 TO 100
30 PSET (100,Y),4:FOR J=1 TO 200:NEXT J
40 PRESET STEP(0,0)
50 NEXT Y
```

パレット番号＝４の点が画面上で移動する。

ＰＲＩＮＴ〔＜式＞〕〔，または；＜式＞・・・・〕

式の値や文字列を画面に表示する。コンマ（，）で区切ると１４文字ずつ区切られる。セミコロン（；）で区切ると次の式の値は連続して表示する。式の最後にコンマもセミコロンもなければ、改行する。キーワードＰＲＩＮＴの代わりに疑問符（？）が使える。引用符とセミコロンが隣接するときは、セミコロンを省略できる。数値をＰＲＩＮＴすると、頭に符号の入るスペースと末尾に１個のスペースがとられる。文字列は、引用符（”）で囲むこと。

ＰＲＩＮＴ　ＵＳＩＮＧ＜”フォーマット文”＞；＜式＞〔，または；＜式＞・・・・〕

文字や数値を指定したフォーマットで表示する。「ＰＲＩＮＴ　ＵＳＩＮＧ一覧表」参照。

ＰＲＩＮＴ＃＜ファイル番号＞，〔＜式＞〕〔，または；＜式＞・・・・〕

シーケンシャルファイルにデータを書く。ファイル番号は、ファイルをＯＰＥＮしたとき指定した番号。

ＰＲＩＮＴ＃＜ファイル番号＞，ＵＳＩＮＧ＜”フォーマット文”＞；〔＜式のリスト＞〕

データをフォーマット文に従いシーケンシャルファイルに書く。フォーマット文は、ＰＲＩＮＴ　ＵＳＩＮＧのフォーマットと同じ。

ＰＳＥＴ（Ｗｘ，Ｗｙ）またはＳＴＥＰ（Ｘ，Ｙ）〔，＜パレット番号＞〕

ワールド座標で指定したドットにパレット番号の色を塗る。パレット番号を省略したときにフォアグラウンドカラーになること以外はＰＲＥＳＥＴ文と同じ。
　パレット番号……省略するとフォアグラウンドカラー。
　ＳＴＥＰ……最終参照座標（ＬＰ）からの相対座標。

```
PUT 〔#〕<ファイル番号>〔,<式>〕
```

バッファからファイルへデータを書き出す。ファイルがディスクとそれ以外では、機能が異なる。ディスクファイルかディスクファイルでないかは、OPEN文で指定したデバイス名で決まる。

ファイル番号 …… OPEN文で指定したファイル番号を指定する。
ディスクファイルのとき …… ランダムバッファの256バイトの内容を1レコードとしてランダムファイルに書き出す。式を省略すると最後にGET、PUTした次のレコード番号として書き出す。
ディスクファイルでないとき …… カセットテープなどのデバイスへデータを書き出す。式は書き出す文字数。式を0と指定するか、省略すると256になる。

```
PUT〔@〕(Sx,Sy),<配列変数名>〔(<要素>)〕
　　　　　または,KANJI(<漢字コード>)〔,<条件>〕
　　　　　　　　〔,<フォアグラウンドカラー>,<バックグラウンドカラー>〕
```

GET@文で配列に読みこんだパターンまたは漢字文字を指定したスクリーン座標にセットする。GET@文参照。

Sx,Sy …… スクリーン座標にセットする四角形の左上点の座標。
配列変数名 …… GET@文で読み込んだ配列変数名。
要素 …… 配列の何番目から表示するかを指定。GET@したときの要素と一致させること。省略したときは、配列の最初から。
漢字コード …… JIS漢字コードを指定。漢字コード表参照。
条件 …… 配列内にあるグラフィックパターンと、すでにある画面のパターンをドットごとに比較し、つぎのような操作をする。ただし、白黒モードのときはドットがあるかないか1ビットずつ操作し、カラーモードのときはドットごとのパレット番号を3ビットずつ操作する。省略するとXOR。

PSET …… 配列内のパターンをそのまま表示する。
PRESET …… 配列内のパターンのNOT(否定)を表示する。白黒モードではパターンが反転する。カラーモードでは、たとえばパレット番号5は2になる。各ドットのパレット番号は、7からそのドットのパレット番号を引いた値になる。
パレット3ビット101(5)のNOTは、010(2=7−5)
OR …… 配列内のパターンと画面のパターンをドットごとにOR(論理和)して表示する。
AND …… 配列内のパターンと画面のパターンをドットごとにAND(論理積)して表示する。
XOR …… 配列内のパターンと画面のパターンをドットごとにXOR(排他的論理和)して表示する。

フォアグラウンドカラー・バックグラウンドカラー …… 白黒モードで配列に読み込まれたパターンに、カラーモードで色をつける。フォアグラウンドカラーは白に対する色、バックグラウンドカラーは黒に対する色の指定。色の指定はパレット番号。指定するときは必ず両方指定すること。漢字パターンは白黒モードで入っている。

PUT@(100,100),KANJI(&H442B) …… "朝"を描く。

```
RANDOMIZE〔<式>〕
```

式によって新しい乱数の種(Seed)を与え、乱数系列を変更する。式の値は、−32768〜32767。
式を省略すると画面に

```
Random number seed (-32768 to 32767)?
```

というメッセージを表示して入力待ちとなる。

```
READ <変数>〔,<変数>・・・・〕
```

DATA文のデータを各変数に読み込む。DATA文参照。

ＲＥＭ　〔＜任意の文字列＞〕

プログラム中に注釈をつける。アポストロフィー（’）でも同じ。マルチステートメントにはできない。

```
１０　ＲＥＭ　＊＊ＫＥＩＳＡＮ
２０　’　＊＊ＰＲＩＮＴ＊＊
３０　’ＨＹＯＵＪＩ：ＰＲＩＮＴ　Ａ　……　ＰＲＩＮＴ　Ａも注釈
```

ＲＥＮＵＭ　〔＜新行番号＞〕〔，＜旧行番号＞〕〔，＜間隔＞〕

行番号をつけなおす。ＧＯＴＯ文などの行き先も自動的に変わる。
　新行番号 …… 省略すると１０。
　旧行番号 …… この行番号以降をつけなおす。省略するとプログラムの最初の行番号。
　間隔 ………… 省略すると１０。

ＲＥＮＵＭ　１０００，１００，２０ ……… 行番号１００～を１０００から２０おきにつけなおす。

ＲＥＳＴＯＲＥ〔＜行番号または＊ラベル名＞〕

ＤＡＴＡ文はＲＥＡＤで１回読むと、あとは使えない。それを解除して、ＤＡＴＡ文をはじめから読めるようにする。あるいは指定した行のＤＡＴＡ文から読めるようにする。行番号を省略すると最初のＤＡＴＡ文となる。

```
　１０　ＲＥＡＤ　Ａ，Ｂ，Ｃ ……… Ａ、Ｂ、Ｃにそれぞれ１、６、３が読まれる。
　　　　：
　５０　ＲＥＳＴＯＲＥ　 ……… ３００行のＤＡＴＡを再度、使うことを宣言。
　　　　：
１００　ＲＥＡＤ　Ｄ，Ｅ，Ｆ ……… Ｄ、Ｅ、Ｆにそれぞれ１、６、３が読まれる。
　　　　：
３００　ＤＡＴＡ　１，６，３
```

ＲＥＳＵＭＥ　〔０またはＮＥＸＴまたは＜行番号または＊ラベル名＞〕

ＯＮ　ＥＲＲＯＲ　ＧＯＴＯ文で指定したエラー処理ルーチンの出口。
　０ ……………… エラー文へ戻る。
　ＮＥＸＴ …… エラー文の次の文へ戻る。
　行番号 ……… 指定した行へ戻る。
　省略したら、０と同じ。

ＲＥＴＵＲＮ〔＜行番号または＊ラベル名＞〕

ＧＯＳＵＢ文で呼び出したサブルーチンの出口。行番号を省略するとＧＯＳＵＢ文の次の文へ戻る。

```
　５０　ＧＯＳＵＢ　２００
　６０　ＰＲＩＮＴ　Ａ
　　　　：　　　　　　　　　　　２００行～３００行がサブルーチン
２００　Ａ＝２５＊Ｘ
３００　ＲＥＴＵＲＮ
```

ＲＯＬＬ　＜ドット数＞　（８８はディスクのみ）

グラフィック画面を指定したドット数だけスクロールアップ（画面がせりあがる）する。
　ドット数 …… ６４０×２００ドットの画面のときは１～１９９、６４０×４００ドットのときは１～３９９。
　　（８８）１～１９７の範囲。

RSET　＜文字変数＞＝＜文字型データ＞

ランダムバッファの文字変数へ文字型データを右づめで移す。ＬＳＥＴ文参照。

RUN　〔＜行番号または＊ラベル名＞〕

プログラムを実行する。行番号を指定すると、そこから実行する。

RUN＜”デバイス名：ファイル名”＞〔，R〕

デバイス名で指定した装置からプログラムファイルをメモリーにロードし、実行する。デバイス名を省略するとドライブ１番。Ｒオプションを省略すると、すでにＯＰＥＮされているファイルをＣＬＯＳＥしてから実行し、Ｒオプションをつけると、ＣＬＯＳＥしないで実行する。

RUN”2：PRG”

SAVE＜”デバイス名：ファイル名”＞〔，AまたはP〕

デバイス名で指定した装置へプログラムをセーブし、ファイル名を付ける。デバイス名を省略するとドライブ１番。

- A ……………… アスキー形式でセーブする。ＭＥＲＧＥ文参照。
- P ……………… 暗号化されたバイナリー形式でセーブする。ＰオプションでＳＡＶＥすると、プログラの内容の表示や変更はできない。
- 省略 ……………… バイナリー形式でセーブする。

SAVE”2：ABC”，P

SCREEN〔＜画面モード＞〕〔，＜画面スイッチ＞〕〔，＜アクティブページ＞〕〔，＜ディスプレーページ＞〕

グラフィック画面のモードを設定する。

画面モード ……… 画面モードを指定する。電源投入時は０。

値	(98)			(88)		
	画面モード	分解能	画面数	画面モード	分解能	画面数
0	カラーモード	640 × 200	2	カラーモード	640 × 200	1
1	白黒モード	640 × 200	6	白黒モード	640 × 200	3
2	高分解能白黒モード	640 × 400	3	高分解能白黒モード	640 × 400	1
3	高分解能カラーモード	640 × 400	1			

画面スイッチ …… グラフィックマスクと高速書き込みを次の値で指定する。電源投入時は０。

値	(98)・(88)	
	グラフィックマスク	高速書き込み
0	OFF	OFF
1	OFF	ON
2	ON	OFF
3	ON	ON

グラフィックマスク	ON	グラフィック画面を一時的に消去する。
	OFF	一時的に消去した画面をもとに戻す。
高速書き込み	ON	グラフィックＲＡＭへの高速書き込み。ただし画面がちらつく。
	OFF	高速書き込みをしない。

アクティブページ
書き込みむページを指定する。
電源投入時は0。

値	書き込むページ	
	(98)	(88)
0	1	1
1	2	2
2	3	3
3	4	
4	5	
5	6	

ディスプレーページ
表示するページを指定する。複数ページのときは、合成して表示する。電源投入時は1。(88)は白黒モードのときだけ有効。

値	(98)		(88)	
	表示ページ	画面モード	表示ページ	画面モード
0	表示しない	全画面モード	表示しない	1
1	1	全画面モード	1	1
2	2	0,1,2	2	1
3	1,2	1,2	1,2	1
4	3	1,2	3	1
5	1,3	1,2	1,3	1
6	2,3	1,2	2,3	1
7	1,2,3	1,2	1,2,3	1
8	表示しない	全画面モード		
9	4	1		
10	5	1		
11	4,5	1		
12	6	1		
13	4,6	1		
14	5,6	1		
15	4,5,6	1		

```
10 SCREEN 0,0
20 CLS 2
30 CIRCLE
    (50,100),50,4
40 PAINT
    (50,100),2,4
50 SCREEN 0,3
60 CIRCLE
    (80,120),50,5
70 PAINT (80,120),3,5
80 SCREEN 0,0
```

SET <ドライブ番号>または#<ファイル番号>または<"ファイル名">,<"属性文字">

ドライブ番号、ファイル番号、ファイルに属性をつける。　*(ディスク)
属性文字
R ……… リード・アフターライト。書き込み検査のため、書いたあと読む。
P ……… ライト・プロテクト。書き込み禁止。
R,P以外 …… 属性を解除する。

SET 1,"R" ……… 1番のドライブへの出力はR検査をする。
SET #1,"R" ……… ファイル1が開かれている間、そこへの出力はRになる。
SET "ABC","P" …ファイル"ABC"をRead onlyにする。
SET 1,"" ……… ドライブ1の属性を解除。

STOP

ブレークメッセージを表示してプログラムの実行を一時停止する。END文と違い使用中のファイルはCLOSEしない。デバッグ中にプログラムの途中で止め、ダイレクトモードで中間結果を調べたり、変更したりできる。CONT命令またはGOTO行番号で再開できる。プログラムを変えるとCONT命令でプログラムの再開はできない。

STOP ONまたはOFFまたはSTOP

STOPキーによる割り込みを許可・禁止または一時停止する。ON STOP GOSUB文と対で使う。
ON ……… 割り込みを許可する。
OFF …… 割り込みを禁止する。
STOP … 割り込みを一時停止する。一時停止の状態で発生した割り込みは、STOP ON文を実行すれば有効になる。

```
ＳＷＡＰ　＜変数＞，＜変数＞
```

２つの変数の値を入れかえる。変数の型は一致していること。

```
１０　Ａ＝３：Ｂ＝４
２０　ＳＷＡＰ　Ａ，Ｂ………… Ａ＝４、Ｂ＝３になる。
```

```
（９８）ＴＥＲＭ”〔ＣＯＭ：〕〔＜パリティー＞〔＜データビット長＞〔＜ストップビット長＞
　　　　〔＜Ｘパラメーター＞　〔＜Ｓパラメーター＞　〔＜ＤＥＬコード受信処理＞
　　　　〔＜リターンキーの送信処理＞　〔＜ＣＲコードの受信処理＞
　　　　〔＜日本語シフトコード＞〕〕〕〕〕〕〕〕〕”
　　　　〔，〔＜通信方式＞〕〔，＜スタック長＞〕〕

（８８）ＴＥＲＭ”〔ＣＯＭ：〕〔＜パリティー＞〔＜データビット長＞〔＜ストップビット長＞
　　　　〔＜Ｘパラメーター＞　〔＜Ｓパラメーター＞〕〕〕〕〕”
　　　　〔，〔＜通信方式＞〕〔，＜スタック長＞〕〕
```

ターミナルモードにする。「通信機能」参照。

パラメーターを空白で指定したり、省略した場合、メモリースイッチ（（８８）はディップスイッチ）で設定した仕様になる。

パリティー	パリティーチェックの指定	Ｅ……偶数パリティーチェック Ｏ……奇数パリティーチェック Ｎ……パリティーチェックなし
データビット長	データビットの長さ	７……７ビット（パリティーはＯかＥ） ８……８ビット（パリティーはＮ）
ストップビット長	ストップビットの長さ	１……１ビット ２……１．５ビット ３……２ビット
Ｘパラメーター	Ｘパラメーターの有効、無効	Ｘ……フロー制御をする Ｎ……フロー制御をしない
Ｓパラメーター	Ｓパラメーターの有効、無効	Ｓ……シフトコードを制御する Ｎ……シフトコードを制御しない
ＤＥＬコード受信処理	ＤＥＬコードの変換	Ｂ……ＢＳコードとして扱う Ｎ……ＮＵＬコードとして扱う
リターンキーの送信処理	リターンキーによる送信コード	Ｃ……ＣＲコードを送信 Ｌ……ＣＲ＋ＬＦコードを送信
ＣＲコードの受信処理	ＣＲコードによる動作	Ｃ……ＣＲコードの受信で復帰と改行動作をする Ｌ……ＣＲコードで復帰動作だけする
日本語シフトコード	日本語文字列の開始、終了	Ｐ……開始　＆Ｈ１Ｂ４Ｂ　終了　＆Ｈ１Ｂ４８ Ｉ……開始　＆Ｈ１Ａ７０　終了　＆Ｈ１Ａ７１
通信方式	通信モード	Ｆ……全二重 Ｈ……半二重
スタック長	リモートＢＡＳＩＣプロトコルを実行するときに使う利用者スタックの長さ 省略すると１０２４	

TIME$　ONまたはOFFまたはSTOP

リアルタイムタイマー（TIME$）による割り込みを許可・禁止または一時停止する。ON　TIME$文と対で使う。
ON ……… 割り込みを許可する。
OFF …… 割り込みを禁止する。
STOP … 割り込みを一時停止する。この間に発生した割り込みはTIME$　ON文を実行したとき有効になる。

TROFF

トレースモードを解除する。次項参照。

TRON

トレースモードで、プログラムの実行を追跡する。実行したプログラムの行番号を画面表示する。

```
  10  TRON
  20  FOR  K=5  TO  7
  30  PRINT  K
  40  NEXT  K
RUN
〔20〕〔30〕5 ……… ループ1回目、5がPRINTされる。
〔40〕〔30〕6 ……… ループ2回目、6がPRINTされる。
〔40〕〔30〕7 ……… ループ3回目、7がPRINTされる。
〔40〕
```

10行のTORNを実行すると、以後はトレースモード。TROFFで解除する。

VIEW（Sx1，Sy1）－（Sx2，Sy2）〔，<領域色>〕〔，<境界色>〕

ディスプレー画面上にビューポートを設定する。座標はオリジナルスクリーン座標で指定する。WINDOW文で指定したウインドー内の図形はビューポート内に表示される。ビューポートの設定を変えるだけで図形を縮小したり、描く位置を変えられる。ただし、図形を描いたあとでビューポートを変えても、図形は変化しない。
境界色を指定したとき、CLS文でビューポート内をクリアーしても境界枠はのこる。

（Sx1，Sy1）…… ビューポートの左上点の座標をオリジナルスクリーン座標で指定する。
（Sx2，Sy2）…… ビューポートの右下点の座標をオリジナルスクリーン座標で指定する。
領域色 ……………… ビューポート内を塗りつぶす。パレット番号を指定。
境界色 ……………… ビューポートの境界枠を書く。パレット番号を指定。

```
100  SCREEN  0，0
110  WINDOW  （0，0）－（639，199）
120  VIEW  （0，0）－（639，199）
130  CLS  2
140  FOR  J=1  TO  5
150     VIEW  （J*20，J*10）－（639－J*20，199－J*10），，4
160     GOSUB  *EN
170  NEXT  J
180  END
190  *EN
200  CIRCLE（320，100），90，3
210  RETURN
```

上のプログラムを動かすと、5つの同心円を描く。緑色の枠は、円を描くときのビューポートを示す。
このようにビューポートを変えるだけで、同じCIRCLE文でも違う大きさの円を描ける。

WAIT <ポート番号>，<マスク>〔，<論理スイッチ>〕

入力ポートをモニターする。次の論理演算の結果が真（０以外）なら次の文を実行し、為（０）なら結果が真になるまでモニターする。

（ポートから読み込んだデータ　XOR　論理スイッチ）AND（マスク）

論理スイッチは０をひろうか、１をひろうかを決める。論理スイッチを＆Ｈ０にすると、値が１のビットは１としてひろいだし、０は０のまま。＆ＨＦＦにすると値が０のビットは１としてひろいだし、１は０にする。マスクは８ビットのうちどのビットをひろいだすか指定する。

（９８）
キーボードからＡのキーを押したら＊＊＊を表示する
Ｉ／Ｏポート参照

```
10  WAIT  &HE2,2,255
20  PRINT  "***"
```

```
なにも押さない状態 …… 255    11111111
論理スイッチ ………… 255    11111111
--------------------------------------
                 (XOR)    00000000
マスク ……………………   2    00000010
--------------------------------------
                 (AND)    00000000 …0
                                 (為)
Aのキーを押した状態 … 253    11111101
論理スイッチ ………… 255    11111111
--------------------------------------
                 (XOR)    00000010
マスク ……………………   2    00000010
--------------------------------------
                 (AND)    00000010 …2
                                 (真)
```

WEND　（８８はディスクのみ）

ＷＨＩＬＥ～ＷＥＮＤと対で使う。次項参照。

WHILE <条件式>　（８８はディスクのみ）

ＷＥＮＤまでの文を、条件式が満たされているあいだ繰り返し実行する。条件式が偽になるとＷＥＮＤの次の文から実行する。最初から偽のときは、ＷＥＮＤの次の文から実行する。

```
10  WHILE  A=0  ──┐
20  WHILE  B>5 ─┐ │
 :              │ │
70  WEND  ──────┘ │
 :                │
90  WEND  ────────┘
```

WIDTH 〔<水平の桁数>〕〔，<垂直の行数>〕

画面に表示する文字の桁数、行数を指定する。
　桁数 …… ４０または８０
　行数 …… ２０または２５

WIDTH<"デバイス名：">，<サイズ>

指定したデバイスに対してバッファサイズを設定する。コミュニケーションポートと、プリンターのみ有効。
　サイズ ………… ０～２５５（０は２５６と解釈）

```
WIDTH#<ファイル番号>,<サイズ>
```

ファイル番号で割り当てられているバッファのサイズを設定する。コミュニケーションポートと、プリンターのみ有効。
　サイズ ………… 0～255（0は256と解釈）

```
WIDTH LPRINT <文字数>
```

プリンターの1行に印字できる文字数を設定する。初期値は80。
　文字数 … 0～255（0は256と解釈）

```
WINDOW (Wx1,Wy1)-(Wx2,Wy2)
```

ビューポートに表示するワールド座標系の領域（ウインドー）を設定する。
　(Wx1,Wy1) …… ワールド座標の左上点の座標。
　(Wx2,Wy2) …… ワールド座標の右下点の座標。

```
10 SCREEN 0,0:CLS 2
20 WINDOW(-200,-200)-(200,200)
30 CIRCLE(0,0),100
40 WINDOW(0,0)-(400,400):CLS 2
50 CIRCLE(200,200),100
```

```
WRITE 〔<式>〔 ,または ; <式>・・・・〕〕
```
（88はディスクのみ）

式の値や文字列をコンマ（,）で区切って表示する。文字列は引用符（"）で囲んで出力する。式と式の間はコンマ（,）かセミコロン（;）で区切るが、結果は同じ。式を省略すると、Missing operandエラーになる。（88）式を省略すると、1行空白を表示。

```
10 A$="ABC":B=123:C=-34
20 WRITE A$,B,C                実行結果は"ABC",123,-34
```

```
WRITE#<ファイル番号>,〔<式>〔,または;<式>・・〕〕
```
（88はディスクのみ）

シーケンシャルファイルへデータを書く。式と式の間はコンマ（,）で区切り、文字列は引用符（"）で囲んで出力する。

４．PRINT USING　一覧表

左づめで表示	！	与えられた文字列の最初の１文字だけをプリントする １０　Ａ＄＝”ＡＢＣ”：Ｂ＄＝”ＤＥＦ” ２０　ＰＲＩＮＴ　ＵＳＩＮＧ　”！”；Ａ＄；Ｂ＄ ＡＤ
	＆　＆	文字データの桁数を指定　空白の個数＋２が桁数 １０　Ａ＄＝”ＡＢＣ”：Ｂ＄＝”ＤＥＦ” ２０　ＰＲＩＮＴ　ＵＳＩＮＧ　”＆　＆　　＆　＆”；Ａ＄；Ｂ＄ ＡＢＣ　ＤＥＦ （９８）漢字の場合、漢字文字数×２＋２（前後のシフトコード）が桁数 １０　Ｃ＄＝”朝日”：Ｄ＄＝”新聞” ２０　ＰＲＩＮＴ　ＵＳＩＮＧ　”＆←空白４個→＆＆←空白４個→＆”；Ｃ＄；Ｄ＄ 朝日新聞
	＠	与えられた文字列に置き換えてプリントする １０　Ａ＄＝”３００” ２０　ＰＲＩＮＴ　ＵＳＩＮＧ　”ＣＯＦＦＥＥ　＝　＠　ＹＥＮ”；Ａ＄ ＣＯＦＦＥＥ　＝　３００　ＹＥＮ （９８） １０　Ｂ＄＝”紅茶”：Ｃ＄＝”３５０”：Ｄ＄＝”円” ２０　ＰＲＩＮＴ　ＵＳＩＮＧ　”＠　＝　＠＠”；Ｂ＄；Ｃ＄；Ｄ＄ 紅茶　＝　３５０円
	＿	＿（アンダーバー）に続く１文字を書式指定文字（＠や＃）とせず単に文字としてプリントする １０　ＰＲＩＮＴ　ＵＳＩＮＧ　”＃＃．＃＃＿＠”；３．２６ 　３．２６＠
右づめで表示	＃	数字の桁数を指定 １０　Ａ＝１２３ ２０　ＰＲＩＮＴ　ＵＳＩＮＧ　”＃＃＃＃”；Ａ 　１２３ ３０　ＰＲＩＮＴ　ＵＳＩＮＧ　”ﾀｲｼﾞｭｳ＃＃＃．＃ｋｇ”；５６．７ ﾀｲｼﾞｭｳ　５６．７ｋｇ
	．	＃と＃の間へ．（ピリオド）をうつと、そこが小数点 １０　Ａ＝１２３．４５６ ２０　ＰＲＩＮＴ　ＵＳＩＮＧ　”＃＃＃＃．＃＃＃”；Ａ 　１２３．４５６ ３０　ＰＲＩＮＴ　ＵＳＩＮＧ　”＃＃＃＃．＃＃”；Ａ 　１２３．４６
	＋	正の符号をつける。数値が負のときは負の符号がつく ＰＲＩＮＴ　ＵＳＩＮＧ　”＋＃＃．＃＃”；７７．８ ＋７７．８０ ＰＲＩＮＴ　ＵＳＩＮＧ　”＃＃．＃＃＋”；－１２．３４ １２．３４－
	－	負の符号をつける。数値が正のときは正の符号はつかない ＰＲＩＮＴ　ＵＳＩＮＧ　”＃＃．＃＃－”；－６８．９５ ６８．９５－ ＰＲＩＮＴ　ＵＳＩＮＧ　”＃＃．＃＃－”；１２．４５ １２．４５ ＰＲＩＮＴ　ＵＳＩＮＧ　”＃＃．＃＃”；－５．６７ －５．６７

<table>
<tr><td rowspan="5">右づめで表示</td><td>＊＊</td><td>頭部の余白を＊でうめる
ＰＲＩＮＴ　ＵＳＩＮＧ　”＊＊＃＃＃．＃＃”；１２．３
＊＊＊１２．３０
ＰＲＩＮＴ　ＵＳＩＮＧ　”＊＊＃＃＃．＃円”；２．５４
＊＊＊＊２．５円</td></tr>
<tr><td>￥￥</td><td>数値の頭に￥をつける
ＰＲＩＮＴ　ＵＳＩＮＧ　”￥￥＃＃＃＃＃”；２０００
　　　￥２０００</td></tr>
<tr><td>＊＊￥</td><td>前の２つの合成
ＰＲＩＮＴ　ＵＳＩＮＧ　”＊＊￥＃＃＃＃＃”；１２３４
＊＊＊￥１２３４</td></tr>
<tr><td>，</td><td>整数部分を３桁ごとに，（コンマ）で区切る
ＰＲＩＮＴ　ＵＳＩＮＧ　”＃＃＃＃＃＃＃＃＃＃，”；１２３４５６７８９
１２３，４５６，７８９</td></tr>
<tr><td>＾＾＾＾</td><td>指数型表示にする
１０　Ａ＝１２３
２０　ＰＲＩＮＴ　ＵＳＩＮＧ　”＃＃．＃＃＾＾＾＾”；Ａ
　１．２３Ｅ＋０２
３０　ＰＲＩＮＴ　ＵＳＩＮＧ　”＃＃．＃＃＾＾＾＾”；－０．０００００５
－５．００Ｅ－０６</td></tr>
<tr><td>％</td><td colspan="2">数字の桁（＃）が足りないと頭に％が表示される
数値を丸めた結果、桁上がりして桁（＃）が不足すると％が表示される
ＰＲＩＮＴ　ＵＳＩＮＧ　”．＃＃”；０．９９５
％１．００</td></tr>
</table>

ＬＰＲＩＮＴ　ＵＳＩＮＧ　とするとプリンターへ出力する。

(例１)

```
１０　Ａ＄＝”ＡＳＡＨＩＳＨＩＭＢＵＮ”
２０　Ｂ＄＝”ＫＳ”
３０　Ｃ＝１０００
４０　ＬＰＲＩＮＴ　ＵＳＩＮＧ　”＆←　空白１０個　→＆　＆＆　＃＃＃＃円”；Ａ＄，Ｂ＄，Ｃ
```

結果

ＡＳＡＨＩＳＨＩＭＢＵＮ　ＫＳ　１０００円

(例２)
（９８）

```
１０　Ｄ＄＝”東京都中央区”
２０　Ｅ＄＝”築地”
３０　Ｆ＄＝”５－３－２”
４０　ＬＰＲＩＮＴ　ＵＳＩＮＧ”＠　＠　＠”；Ｄ＄；Ｅ＄；Ｆ＄
```

結果

東京都中央区　築地　５－３－２

５．数値関数

ｘは数式、Ｓは文字式とする

関　数	機　　能	備　　考
ＡＢＳ（ｘ）	ｘの絶対値を与える	ＰＲＩＮＴ　ＡＢＳ（－４０）は４０
ＡＴＮ（ｘ）　注 （８８）は （ディスク）	ｘのアークタンジェント 結果はラジアン	ｘはラジアン。ラジアンと度数の関係は １ラジアン＝（１８０／π）度 １度＝π／１８０　ラジアン
ＣＤＢＬ（ｘ）	ｘを倍精度実数型に変換	Ａ＃＝ＣＤＢＬ（Ｂ）
ＣＩＮＴ（ｘ）	ｘを整数型に変換	ｘの値の小数点以下は、四捨五入する －３２７６８≦ｘ≦３２７６７
ＣＯＳ（ｘ）　注	ｘのコサイン	ｘはラジアン
ＣＳＮＧ（ｘ）	ｘを単精度実数型に変換	Ａ＝ＣＳＮＧ（Ｃ＋２）
ＣＶＤ（Ｓ）	８バイトの文字列を倍精度の数値に変換	ランダムファイルから読み込んだ数値は文字型のため数値に変換する １０　ＦＩＥＬＤ＃１，４　ＡＳ　Ａ＄… ２０　ＧＥＴ＃１ ３０　Ａ＝ＣＶＳ（Ａ＄）
ＣＶＳ（Ｓ）	４バイトの文字列を単精度の数値に変換	
ＣＶＩ（Ｓ）	２バイトの文字列を整数の数値に変換	
ＥＸＰ（ｘ）　注	ｅのｘ乗	ＰＲＩＮＴ　ＥＸＰ（１）は ２．７１８２８ Ｘ＝ｅ＾Ｙ　とすればＬＯＧとの関係は Ｙ＝ＬＯＧｅＸ
ＦＩＸ（ｘ）	小数点以下を切り捨てた整数	ＰＲＩＮＴ　ＦＩＸ（４．５６）は４ ＰＲＩＮＴ　ＦＩＸ（－４．５）は－４
ＩＮＴ（ｘ）	ｘを超えない最大の整数	ＰＲＩＮＴ　ＩＮＴ（４．５６）は４ ＰＲＩＮＴ　ＩＮＴ（－４．５）は－５
ＬＯＧ（ｘ）　注	ｘの自然対数	ｘ＞０とする。自然対数を常用対数にする式はＬＯＧ（ｘ）／ＬＯＧ（１０）
ＲＮＤ（ｘ）　注	一様乱数をつくる　０＜乱数＜１ ｘ＞０なら通常の乱数 ｘ＝０なら直前の乱数と同じ ｘを省略するとｘ＞０	乱数系列はＲＡＮＤＯＭＩＺＥ文で変える （８８）ではｘ＜０なら乱数系列を変える
ＳＧＮ（ｘ）	ｘの正負符号を与える	Ａ＝ＳＧＮ（ｘ）　ｘ＞０ならＡ＝１ ｘ＝０ならＡ＝０ ｘ＜０ならＡ＝－１
ＳＩＮ（ｘ）　注	ｘのサイン	ｘはラジアン
ＳＱＲ（ｘ）　注	ｘの平方根	Ａ＝ＳＱＲ（２）　ｘ≧０であること
ＴＡＮ（ｘ）　注	ｘのタンジェント	ｘはラジアン

注ＤＩＳＫ－ＢＡＳＩＣの場合、ｘが倍精度なら関数の値も倍精度

６．ストリング関数

ｘ、ｎ、ｍは数式、Ｓは文字式とする

関　数	機　　能	備　　考
ＡＳＣ（Ｓ）	文字列Ｓの最初の文字のコードを数値で与える	Ａ＄＝”ＡＢＣ” ＰＲＩＮＴ　ＡＳＣ（Ａ＄）は６５ ＢとＣは無視される
ＣＨＲ＄（ｘ）	コードｘに対応する文字を与える	ＰＲＩＮＴ　ＣＨＲ＄（７）は　ＢＥＥＰ ＰＲＩＮＴ　ＣＨＲ＄（８４）は　Ｔ ＰＲＩＮＴ　ＣＨＲ＄（＆ＨＦ１）は　円
ＨＥＸ＄（ｘ）	ｘの１６進数文字列を与える	ＰＲＩＮＴ　ＨＥＸ＄（２５５）はＦＦ
ＩＮＫＥＹ＄	キーボードが押されたかどうか調べる 押されればその文字を与える	１０　Ｘ＄＝ＩＮＫＥＹ＄ ２０　ＩＦ　Ｘ＄＝””　ＴＨＥＮ　１０ ３０　ＰＲＩＮＴ　Ｘ＄
ＩＮＳＴＲ（〔ｎ，〕Ｓ１，Ｓ２）	Ｓ１からＳ２と同じ文字列を探し、合致したら、Ｓ２の最初の文字がＳ１の何文字目かを与える 同じ文字列がみつからなければ０ ｎを指定するとＳ１のｎ文字目から探しはじめる ｎを省略すると１文字目から	Ｘ＄＝”ＡＢＣＡＢＣＤ” Ｙ＄＝”ＣＤ” ＰＲＩＮＴ　ＩＮＳＴＲ（Ｘ＄，”Ｃ”）は　３ Ａ＝ＩＮＳＴＲ（４，Ｘ＄，Ｙ＄） ＰＲＩＮＴ　Ａ　は６
ＬＥＦＴ＄（Ｓ，ｎ）	Ｓの左からｎ番目までの文字列を与える ｎは０～２５５	Ｘ＄＝”ＡＢＣＤＥＦ” ＰＲＩＮＴ　ＬＥＦＴ＄（Ｘ＄，３）はＡＢＣ
ＬＥＮ（Ｓ）	Ｓの文字数を与える	Ｘ＄＝”ＡＢＣＤＥＦ” ＰＲＩＮＴ　ＬＥＮ（Ｘ＄）は　６
ＭＩＤ＄（Ｓ，ｎ，ｍ）	Ｓの左ｎ文字目から長さｍの文字列を与える ＭＩＤ＄は代入文の左辺でも使うことができる Ｓの左ｎ文字目から長さｍに右辺の文字を入れる	１０　Ｘ＄＝”ＡＢＣＤＥＦ” ２０　Ｙ＄＝ＭＩＤ＄（Ｘ＄，３，２） ３０　ＰＲＩＮＴ　Ｙ＄ は　ＣＤ １０　Ａ＄＝”ＡＡＡＡＡ” ２０　ＭＩＤ＄（Ａ＄，２，２）＝”●●” ３０　ＰＲＩＮＴ　Ａ＄ は　Ａ●●ＡＡ
ＭＫＤ＄（ｘ）	倍精度実数ｘを８バイトの文字列に変換する	ＬＳＥＴまたはＲＳＥＴを使い、ランダムバッファに入れる数値はすべて文字列に変換する １０　ＯＰＥＮ”ＦＩＬＥ”　ＡＳ　＃１ ２０　ＦＩＥＬＤ　＃１，２０　ＡＳ　Ｎ＄，４　ＡＳ　Ａ＄ ３０　Ｘ＄＝”ＡＢＣ” ４０　ＬＳＥＴ　Ｎ＄＝Ｘ＄ ５０　Ａ＝３．４５６ ６０　ＲＳＥＴ　Ａ＄＝ＭＫＳ＄（Ａ） ７０　ＰＵＴ　＃１ ８０　ＣＬＯＳＥ　＃１ ９０　ＥＮＤ
ＭＫＳ＄（ｘ）	単精度実数ｘを４バイトの文字列に変換する	
ＭＫＩ＄（ｘ）	整数ｘを２バイトの文字列に変換する	

x、nは数式、Sは文字式とする

関　数	機　　　能	備　　　考
OCT$(x)	数値xの8進数文字列を与える	PRINT OCT$(&H11)は21
RIGHT$ (S,n)	Sの右からn番目までの文字列を与える	X$="ABCDEF" PRINT RIGHT$(X$,3)はDEF
SPACE$ (n)	n個の空白を与える	10 X$="ABC" 20 Y$="DE" 30 C$=X$+SPACE$(7)+Y$ 40 PRINT C$ は ABC△△△△△△△DE (△は空白とする)
STR$(x)	数値xを文字列に変換する	PRINT STR$(123)+"円" は 123円
STRING$ (n,<Sまたはx>)	Sの最初の文字をn個、またはアスキーコードxの文字をn個与える	PRINT STRING$(5,"ABC")は AAAAA PRINT STRING$(5,&HEC)は ●●●●●
VAL(S)	Sを数値に変換する Sの最初の文字が+、−、&、数字でなければ0になる	PRINT VAL("010")は10 PRINT VAL("A10")は0

７．入出力関数

ｐは数式とする

関　　数	機　　　　能	備　　　　考
ＡＴＴＲ＄ （＜ドライブ番号または　＃ファイル番号　またはファイル名＞） （ディスク）	指定したディスクやファイルの属性を与える ”Ｒ　　”……リード・アフター・ライト ”　Ｐ”……ライトプロテクト ”　Ｅ　”……ＳＡＶＥ文でＰオプション指定	ＳＥＴ　１，”Ｐ” ＰＲＩＮＴ　ＡＴＴＲ＄（１） は　△△Ｐ（△は空白とする） ＳＥＴ　１，”Ｒ” ＰＲＩＮＴ　ＡＴＴＲ＄（１） は　Ｒ△△
（片面ディスク） ＤＳＫＩ＄（＜ドライブ番号＞，＜トラック番号＞，＜セクター番号＞） （両面ディスク） ＤＳＫＩ＄（＜ドライブ番号＞，＜サーフェス番号＞，＜トラック番号＞，＜セクター番号＞） （ディスク）	ＯＰＥＮ、ＣＬＯＳＥを必要とせず、ディスクの指定したセクターを直接＃０のバッファに読みこむとともに関数値としても与える。関数値で返すときは２５５文字となる	１０　ＦＩＥＬＤ　＃０，４　ＡＳ　Ａ＄ ２０　Ｘ＄＝ＤＳＫＩ＄（１，０，１） ３０　ＰＲＩＮＴ　Ａ＄ ４０　Ｙ＄＝ＭＩＤ＄（Ｘ＄，１，４） ５０　ＰＲＩＮＴ　Ｙ＄
ＤＳＫＦ（＜ドライブ番号＞〔，ｐ〕） （ディスク）	指定したディスクの情報を与える ｐの値 ０……最大トラック番号 １……１トラック当たりのセクター数 ２……片面（０）か両面（１）かのディスクの区別 ３……１トラック当たりのクラスター数 ４……ボリューム当たりのクラスター数 ５……ディレクトリートラック番号 ６……１クラスター当たりのセクター数 ７……ＦＡＴの最初のセクター番号 ８……ＦＡＴの最後のセクター番号 ９……ＦＡＴの数 10……ディスク属性のあるセクター番号 省略……ディスクの未使用のクラスター数	１０　ＰＲＩＮＴ　ＤＳＫＦ（１） ２０　ＦＯＲ　Ｋ＝０　ＴＯ　１０ ３０　ＰＲＩＮＴ　ＤＳＫＦ（１，Ｋ） ４０　ＮＥＸＴ　Ｋ
ＥＯＦ（＜ファイル番号＞）	シーケンシャルファイルの場合 指定した番号のファイルの終わりを示す 終わりになると－１（真）を与える ＲＳ－２３２Ｃコミュニケーションファイルの場合 バッファが空のとき－１（真）を与える	ファイルの終わりを判定 １０　ＯＰＥＮ　………… ２０　ＩＦ　ＥＯＦ（１）ＴＨＥＮ　６０ ３０　ＩＮＰＵＴ＃１，Ａ＄ ４０　ＰＲＩＮＴ　Ａ＄ ５０　ＧＯＴＯ　２０ ６０　ＣＬＯＳＥ　＃１ ７０　ＥＮＤ

ｘ、ｎ、ｐは数式とする

関　　数	機　　　　能	備　　　　考
ＦＰＯＳ（＜ファイル番号＞）	ディスクの場合 　最後に読み込んだ（または書き出した）セクター番号を与える プリンターの場合 　プリンターのヘッドの位置を与える	プリンターのときはＬＰＯＳと同じ
ＩＮＰＵＴ＄（ｎ〔，〔＃〕ｘ〕）	キーボードまたはファイル番号ｘのファイルから入力したｎ文字分の文字列を与える ただし、ｎは２５５以下	キーボードから入力した文字は画面表示されない。すべてのコントロール文字は、ＣＴＲＬ－Ｃをのぞきそのまま入力される Ｘ＄＝ＩＮＰＵＴ＄（２）など
ＬＯＣ（＜ファイル番号＞）	ランダムファイルの場合 　レコード番号を指定しないＧＥＴまたはＰＵＴを実行したときに使用したレコード番号を与える シーケンシャルファイルの場合 　ファイルが開かれてから読み込んだ（または書き出した）セクター数を与える。ＬＯＦと同じ値になる ＲＳ－２３２Ｃコミュニケーションファイルの場合 　入力バッファにたまっている文字数を与える キーボードファイルの場合 　バッファにたまっている文字数を与える	１０　ＯＰＥＮ　……… ２０　ＦＩＥＬＤ　＃１，４　ＡＳ　Ａ＄ ３０　ＩＦ　ＬＯＣ（１）＝ＬＯＦ（１） 　　　　　　ＴＨＥＮ　７０ ４０　ＧＥＴ　＃１ ５０　ＰＲＩＮＴ　Ａ＄ ６０　ＧＯＴＯ　３０ ７０　ＣＬＯＳＥ　＃１ ８０　ＥＮＤ
ＬＯＦ（＜ファイル番号＞）	ディスクファイルの場合 　そのファイルの大きさをセクター数で与える。ランダムファイルの場合最大のレコード番号 ＲＳ－２３２Ｃコミュニケーションファイルの場合 　バッファの残りバイト数を与える	１０　ＯＰＥＮ　……… ２０　ＦＯＲ　Ｋ＝１　ＴＯ　１００ ３０　ＰＲＩＮＴ　＃１，”ＡＡＡＡ” ４０　ＮＥＸＴ　Ｋ ５０　ＰＲＩＮＴ　ＬＯＦ（１） ６０　ＣＬＯＳＥ　＃１ ７０　ＥＮＤ
ＬＰＯＳ（ｘ）	プリンターのヘッドの位置を与える ｘはダミー変数	１行６０文字で改行 ＩＦ　ＬＰＯＳ（１）＝　６０ ＴＨＥＮ　ＬＰＲＩＮＴ
ＰＥＮ（ｐ）	ライトペンからの情報を与える ｐの値 　０ …… ペンを押したとき真（－１）押さないとき偽（０）を与える。真になるとＯＮ　ＰＥＮ　ＧＯＳＵＢ文で定義した割り込み処理ルーチンに入るまで真のまま 　１ …… ペンを押している間キャラクター座標の水平座標を与える 　２ …… ペンを押している間キャラクター座標の垂直座標を与える	

8．画面・グラフィック関数

ｘ、ｙ、ｐは数式とする

関　　数	機　　　　能	備　　　　考
ＣＳＲＬＩＮ	画面上のカーソルの縦の位置を与える	１０　ＬＯＣＡＴＥ　５，６ ２０　Ｙ＝ＣＳＲＬＩＮ　　Ｙの値は６
ＭＡＰ（ｘ，ｐ）	ワールド座標をスクリーン座標に変換またはスクリーン座標をワールド座標に変換する ｐの値 ０ ‥‥ Ｘ方向のワールド座標ｘを対応するスクリーン座標に変換する １ ‥‥ Ｙ方向のワールド座標ｘを対応するスクリーン座標に変換する ２ ‥‥ Ｘ方向のスクリーン座標ｘを対応するワールド座標に変換する ３ ‥‥ Ｙ方向のスクリーン座標ｘを対応するワールド座標に変換する	１０　ＷＩＮＤＯＷ（－８０，－５０）－（０，０） ２０　ＰＲＩＮＴ　ＭＡＰ（－８０，０） ３０　ＰＲＩＮＴ　ＭＡＰ（－５０，１） ４０　ＰＲＩＮＴ　ＭＡＰ（０，２） ５０　ＰＲＩＮＴ　ＭＡＰ（０，３） 実行結果は ０ ０ －８０ －５０
ＰＯＩＮＴ（ｐ）	グラフィック画面のＬＰをワールド座標またはスクリーン座標で与える ｐの値 ０ ‥‥ Ｘ座標をワールド座標で与える １ ‥‥ Ｙ座標をワールド座標で与える ２ ‥‥ Ｘ座標をスクリーン座標で与える ３ ‥‥ Ｙ座標をスクリーン座標で与える	１０　ＣＬＳ　３ ２０　ＷＩＮＤＯＷ（－１００，－１００）－（１００，１００） ３０　ＬＩＮＥ（１０，１０）－（２０，２０），４，Ｂ ４０　ＰＲＩＮＴ　ＰＯＩＮＴ（０）； ５０　ＰＲＩＮＴ　ＰＯＩＮＴ（１）； ６０　ＰＲＩＮＴ　ＰＯＩＮＴ（２）； ７０　ＰＲＩＮＴ　ＰＯＩＮＴ（３）； ２０　２０　３８３　１１９
ＰＯＩＮＴ（ｘ，ｙ）	スクリーン座標（ｘ，ｙ）のドットの色をパレット番号で与える	１０　ＰＲＩＮＴ　ＰＯＩＮＴ（８，５）
ＰＯＳ（０）	画面上のカーソルの横の位置を与える	１０　ＬＯＣＡＴＥ　５，６ ２０　Ｘ＝ＰＯＳ（０）　Ｘの値は５
ＶＩＥＷ（ｐ）	ビューポートの位置をオリジナルスクリーン座標で与える ｐの値 ０ ‥‥ 左上点のＸ座標 １ ‥‥ 左上点のＹ座標 ２ ‥‥ 右下点のＸ座標 ３ ‥‥ 右下点のＹ座標	１０　ＷＩＮＤＯＷ（０，０）－（１００，５０） ２０　ＶＩＥＷ（５０，５０）－（１５０，１００） ３０　ＰＲＩＮＴ　ＶＩＥＷ（０）； ４０　ＰＲＩＮＴ　ＶＩＥＷ（１）； ５０　ＰＲＩＮＴ　ＶＩＥＷ（２）； ６０　ＰＲＩＮＴ　ＶＩＥＷ（３） ７０　ＰＲＩＮＴ　ＷＩＮＤＯＷ（０）； ８０　ＰＲＩＮＴ　ＷＩＮＤＯＷ（１）； ９０　ＰＲＩＮＴ　ＷＩＮＤＯＷ（２）； １００　ＰＲＩＮＴ　ＷＩＮＤＯＷ（３） は ５０　５０　１５０　１００ ０　０　１００　５０
ＷＩＮＤＯＷ（ｐ）	ウインドーの位置をワールド座標で与える ｐの値 ０ ‥‥ 左上点のＸ座標 １ ‥‥ 左上点のＹ座標 ２ ‥‥ 右下点のＸ座標 ３ ‥‥ 右下点のＹ座標	

9．その他の関数

ｘ、ｐ、ｎ、ｍは数式

関　数	機　能	備　考
ＤＡＴＥ＄	内蔵クロックの日付。電源を切ってもバッテリーバックアップで、日付が更新される	ＰＲＩＮＴ　ＤＡＴＥ＄ 日付のセットは ＤＡＴＥ＄は”ＹＹ／ＭＭ／ＤＤ”の形式 ＤＡＴＥ＄＝”８２／０８／０１”など
ＥＲＬ	エラーが起きた行番号を与える	１０　ＯＮ　ＥＲＲＯＲ　ＧＯＴＯ　５０ ２０　ＩＮＰＵＴ　Ａ ３０　ＰＲＩＮＴ　１０／Ａ ４０　ＧＯＴＯ　２０ ５０　ＩＦ　ＥＲＲ＝１１　ＡＮＤ 　　　　ＥＲＬ＝３０ 　　　ＴＨＥＮ　ＲＥＳＵＭＥ　２０ ６０　ＥＮＤ
ＥＲＲ	エラーのコードを与える	
ＦＲＥ（ｐ）	ユーザー用メモリーエリアの空きバイト数を与える ｐの値 ０ …… 変数領域の空きバイト数 １ …… テキスト領域の空きバイト数 ２ …… 変数領域の空きバイト数＋テキスト領域の空きバイト数 ３ …… 配列変数領域の空きバイト（９８）のみ	ＰＲＩＮＴ　ＦＲＥ（０） ｐは数式でもよいが整数型であること
ＩＮＰ（ｘ）	Ｉ／Ｏアドレスｘのポートからよんだデータの値を与える	キーボードで押したキーの値をＡに与える （９８）Ａ＝ＩＮＰ（＆ＨＥ３） （８８）Ａ＝ＩＮＰ（３）
ＰＥＥＫ（ｘ）	メモリーｘ番地の内容を与える （９８）ｘはセグメントのオフセット	Ａ＝ＰＥＥＫ（＆Ｈ１０５６） ＰＯＫＥの逆
ＳＥＡＲＣＨ（配列変数名，ｘ〔，ｎ〕〔，ｍ〕） （ディスク）	整数型一次元配列のｎ番めからｍずつ増やしてｘを捜し、最初に見つかったときの要素番号を与える。見つからないとき、−１を与える ｎを省略すると配列の最初から、ｍを省略すると１	Ｋ＝ＳＥＡＲＣＨ（Ａ％，１００）は、Ａ％の配列の最初から１ずつ増やしながら１００を捜し、最初に見つかったときの要素番号をＫに入れる
ＳＰＣ（ｎ）	ＰＲＩＮＴ文中で使い、ｎ個の空白を与える	ＰＲＩＮＴ　”Ａ”；ＳＰＣ（３）；”Ｂ” Ａ△△△Ｂ（△は空白とする）
ＴＡＢ（ｎ）	ＰＲＩＮＴ文中で使い、左端からｎ字までスキップする。現在のカーソルの位置＞ｎならば無効	１０　ＰＲＩＮＴ　”ＡＡＡ”； ２０　ＰＲＩＮＴ　ＴＡＢ（５）；”ＢＢＢ” ＡＡＡ△△ＢＢＢ
ＴＩＭＥ＄	内蔵クロックの時刻。電源を切ってもバッテリーバックアップで、時刻が更新される	ＰＲＩＮＴ　ＴＩＭＥ＄ 時刻のセットは ＴＩＭＥ＄は”ＨＨ：ＭＭ：ＳＳ”の形式 ＴＩＭＥ＄＝”２３：００：１５”など

pは数式

関　数	機　能	備　考
USR<N> (n)	ユーザーが定義したN番目の機械語ルーチンへとぶ	Nは0から9まで。省略すると0 nは引数 DEF USR文参照
(98) VARPTR (<変数名または#ファイル番号>〔,p〕)	変数の割り当てられている番地を与える。ファイル番号を指定した場合には指定したファイル番号に与えられた入出力バッファの開始番地を与える pの値 0 ……オフセット 0以外…セグメントベース番地 省略すると0	10 K%=256 20 A=VARPTR(K%,1) 30 B=VARPTR(K%,0) 40 DEF SEG=A 50 PRINT PEEK(B+1) 60 PRINT PEEK(B) 1 0
(88) VARPTR (<変数名または#ファイル番号>)	変数の割り当てられている番地を与える。ファイル番号を指定した場合には指定したファイル番号に与えられた入出力バッファの開始番地を与える	10 K%=256 20 A=VARPTR(K%) 30 PRINT PEEK(A+1) 40 PRINT PEEK(A) 1 0

10．9801　日本語処理のステートメントと関数

《ステートメント》

KINPUT＜変数＞

キーボードから日本語文字を入力する。変数は文字変数で、添字付きでもよい。日本語文字の入力は、JIS漢字コード4桁で入力する。入力するごとに日本語1文字が表示される。入力中のカーソル移動やコードの誤りはエラーとなり、再入力が必要。このステートメントを使えば【CTRL】＋【XFER】で日本語入力モードに切り換える必要はない。

```
10　KINPUT　K$
20　PRINT　　K$
```

《関数（ディスク）》

以下の説明で「文字」と「漢字」は次の意味をもつ。
「文字」は1バイトで表現する英数字、カナなど（キャラクターコード表参照）
「漢字」は2バイトで表現する日本語文字（JIS漢字コード表参照）
備考欄の小文字の文字定数は「文字」、大文字の文字定数は「漢字」とする。

n、pは数式、Sは文字式

関　数	機　能	備　考
AKCNV$（S）	文字列Sの中の文字を漢字に変換する シフトコードも入る	K$＝”ABC 朝日” PRINT　AKCNV$（K$）は ＡＢＣ朝日
JIS$（S）	文字列Sの中の最初の2バイトをJIS漢字コードで与える	PRINT　JIS$（KMID$（”朝日”，2，1））　は　442B
KACNV$（S）	文字列Sの中の漢字を文字に変換する シフトコードは取り除く	K$＝”ＡＢＣDEF ” PRINT　KACNV$（K$）は ABCDEF
KEXT$（S，p）	文字列Sの中から文字または漢字を抜き出す シフトコードは対象外 pの値 0…文字列Sの中の文字を抜き出す 1…文字列Sの中の漢字を抜き出す 文字列Sの中に指定した文字や漢字が無ければ””（ヌルストリング）を与える	K$＝” ABC朝日DEFG新聞” PRINT　KEXT$（K$，0）は ABCDEFG PRINT　KNJ$（”1B4B”）＋KEXT$（K$，1）＋KNJ$（”1B48”）は 朝日新聞 1B4Bは漢字インシフトコード 1B48は漢字アウトシフトコード
KINSTR（〔n，〕S1，S2）	S1からS2と同じ文字列を探し、合致したらS2の最初の文字がS1の何文字目かを与える。漢字シフトコードも文字数に含む 同じ文字列が無ければ0 nを与えると、S1のn文字目から探す ここで扱う文字は「漢字」でも「文字」でもよい	X$＝”朝日新聞” Y$＝”新聞” PRINT　KINSTR（X$，”朝”）は　2 PRINT　KINSTR（X$，Y$）は　4 PRINT　KINSTR（3，X$，”新”）は　4

《関数（ディスク）》

ｎ、ｐは数式、Ｓは文字式

関　数	機　能	備　考
ＫＬＥＮ （Ｓ〔，ｐ〕）	文字列Ｓの中の文字数を与える ｐの値 ０…全体の文字数 シフトコードを含み漢字は２バイトで１文字 １～４の文字数にはシフトコードは含まない １…文字の文字数 ２…漢字の文字数 ３…全角漢字の文字数 ４…半角漢字の文字数 ５…シフトコード数	１０　Ｘ＄＝”abc 新聞” ２０　ＦＯＲ　Ｋ＝０　ＴＯ　５ ３０　ＰＲＩＮＴ　ＫＬＥＮ（Ｘ＄，Ｋ） ４０　ＮＥＸＴ　Ｋ　　は ７ ３ ２ ２ ０ ２ ｐを省略すると、機能は０と同じ
ＫＭＩＤ＄ （Ｓ，ｎ〔，ｍ〕 ）	文字列Ｓの左ｎ文字目から長さｍの文字列を与える ｍを省略したときは、ｎ文字目以降のすべてを与える Ｓ＜ｎのとき””（ヌルストリング）をあたえる シフトコードも漢字１文字と数える ここで扱う文字は「漢字」でも「文字」でもよい	Ｘ＄＝”朝日新聞社” ＰＲＩＮＴ　ＫＭＩＤ＄（Ｘ＄，１，３）は 朝日 ＰＲＩＮＴ　ＫＭＩＤ＄（Ｘ＄，１）は 朝日新聞社
ＫＮＪ＄（Ｓ）	文字列Ｓの最初の４桁を漢字に変換する 文字列は１６進４桁のＪＩＳ漢字コード	Ａ＄＝”1B4B” Ｂ＄＝”442B” Ｃ＄＝”467C” Ｄ＄＝”1B48” ＰＲＩＮＴ　ＫＮＪ＄（Ａ＄）＋ＫＮＪ＄（Ｂ＄）＋ＫＮＪ＄（Ｃ＄）＋ＫＮＪ＄（Ｄ＄）は 朝日
ＫＴＹＰＥ （Ｓ，ｎ）	文字列Ｓの中のｎ番目の文字のタイプを与える ０…文字 １…全角漢字 ２…半角漢字 ３…漢字インシフトコード ４…漢字アウトシフトコード シフトコードも漢字１文字と数える ここで扱う文字は「漢字」でも「文字」でもよい	Ｘ＄＝”朝日新聞” ＰＲＩＮＴ　ＫＴＹＰＥ（Ｘ＄，１）は ３ ＰＲＩＮＴ　ＫＴＹＰＥ（Ｘ＄，３）は １ ＰＲＩＮＴ　ＫＴＹＰＥ（Ｘ＄，６）は ４

11．9801　表示選択方式による日本語入力

日本語入力☆

Ｎ８８—日本語ＢＡＳＩＣを使えば従来のＪＩＳコード入力による漢字入力に加え、表示選択方式による入力ができる。漢字の読みをカナやローマ字で入力すると、読みに対応する漢字の一覧表を表示する。その中から該当する漢字を選べばよい。この方式はスクリーンエディターを使ってＢＡＳＩＣのプログラムを作る場合と、以下の命令文でキーボードからデータを入力する場合に使える。
ＩＮＰＵＴ、ＩＮＰＵＴ　ＷＡＩＴ、ＬＩＮＥ　ＩＮＰＵＴ、ＬＩＮＥ　ＩＮＰＵＴ　ＷＡＩＴ、ＫＩＮＰＵＴ

《日本語入力モード》
表示選択方式による日本語入力を可能にするには、日本語入力モードに切り替える必要がある。次の条件で【ＣＴＲＬ】と【ＸＦＥＲ】を押せば日本語入力モードに切り替わる。再度【ＣＴＲＬ】と【ＸＦＥＲ】を押せば日本語入力モードから抜け出す。

日本語入力モードの条件

画面モード	スクロール最終行	カーソルの位置	ファンクションキーの表示
８０桁２５行	２４行目以前	２４行目以前	無関係
	２５行目	２４行目以前	表示が有る場合に限る
８０桁２０行	１９行目以前	１９行目以前	無関係
	２０行目	１９行目以前	表示が有る場合に限る

日本語入力モードの条件を満たさないで【ＣＴＲＬ】と【ＸＦＥＲ】を押せばＪＩＳコード入力モードに切り替わる。４０桁モードや、８０桁２５行モードでスクロール最終行が２５行目かつファンクションキーの表示がない場合などがこれに相当する。
ＫＩＮＰＵＴ文で漢字データの入力要求を出した場合には、【ＣＴＲＬ】と【ＸＦＥＲ】を押さなくても自動的にモードが切り替わる。日本語入力モードになる条件は同じ。

《入力方式と入力文字の切り替え》
入力方式は、キーボードの文字を直接入力する方式、ローマ字でかな文字を入力する方式、ＪＩＳコードで入力する方式の三種類ある。直接入力方式は、ひらがな、カタカナ、英小文字、英大文字の四種類入力できる。ローマ字入力はローマ字に対応するカタカナとひらがなを入力できる。ＪＩＳコード方式は１６進コード４桁で入力する。

切り替え方法

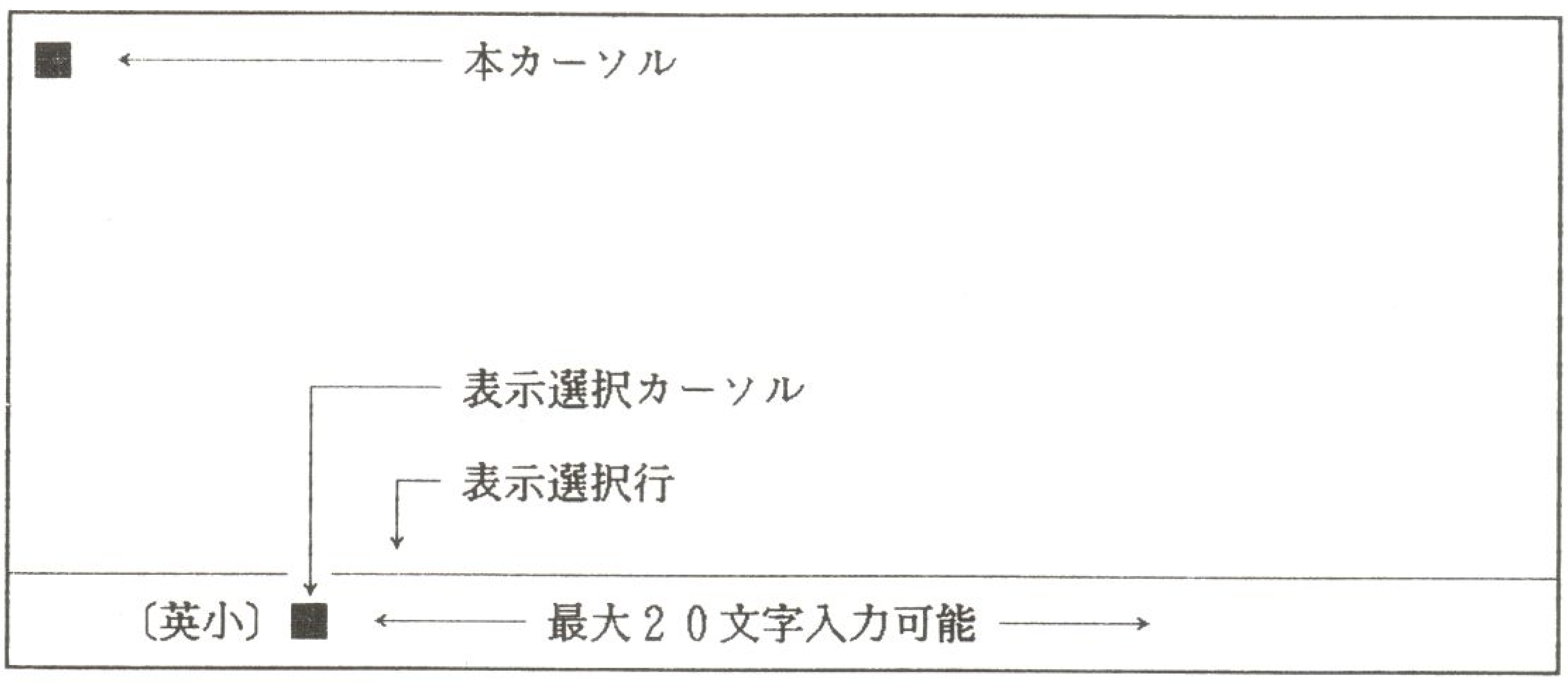
<table>
<tr><td colspan="6">【ＣＴＲＬ】＋【Ｖ】で交互に切り替え</td><td>【ＣＴＲＬ】＋【Ｐ】</td></tr>
<tr><td colspan="4">直　接　入　力　方　式</td><td colspan="2">ローマ字入力方式</td><td>ＪＩＳコード入力方式</td></tr>
<tr><td colspan="2">【カナ】ロック解除</td><td colspan="2">【カナ】ロック</td><td colspan="2">【カナ】ロック解除</td><td rowspan="4">ＪＩＳコード４桁入力</td></tr>
<tr><td colspan="2">英字モード</td><td colspan="2">かなモード</td><td colspan="2">かなモード</td></tr>
<tr><td colspan="2">【ＣＡＰＳ】で交互に切り替え　(注)</td><td colspan="2">【ＣＴＲＬ】＋【Ｑ】で交互に切り替え</td><td colspan="2">【ＣＴＲＬ】＋【Ｑ】で交互に切り替え</td></tr>
<tr><td>英小文字入力</td><td>英大文字入力</td><td>ひらがな入力</td><td>カタカナ入力</td><td>ひらがな入力</td><td>カタカナ入力</td></tr>
</table>

【ＣＴＲＬ】と【Ｐ】を押すとＪＩＳコード入力方式になり、ＪＩＳコード４桁（１６進）を入力すると自動的に元のモードに戻る。入力の前に【ＣＴＲＬ】と【Ｐ】を押せば、ＪＩＳコード入力モードを取消せる。

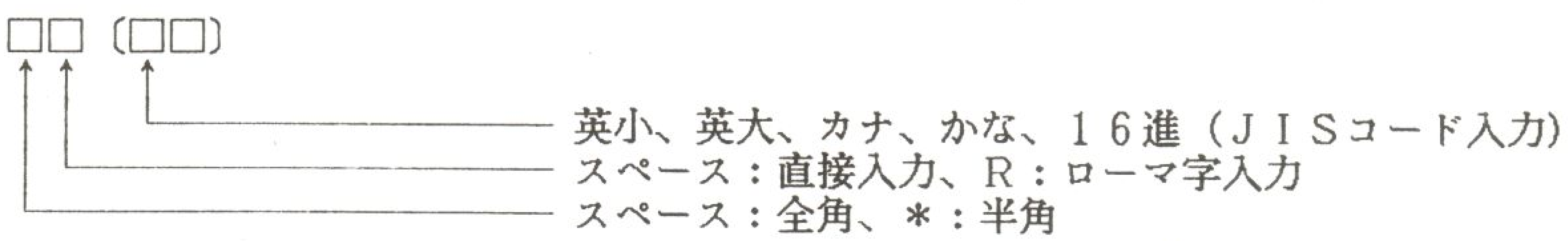
【ＣＴＲＬ】と【Ｔ】を押すと全角文字と半角文字が交互に切り替わる。半角文字は英数字、特殊記号、カタカナだけ表示できる。ひらがなや漢字は全角文字でなければ表示できない。

（注）
【ＣＡＰＳ】ロックの状態で【ＳＨＩＦＴ】キーを押しながら英字を入れると英小文字になる。
【ＣＡＰＳ】ロック解除の状態で【ＳＨＩＦＴ】キーを押しながら英字を入れると英大文字になる。

《日本語入力モードの画面》
日本語入力モードでは下図のように二個のカーソルが表れる。従来の入力位置を示すカーソルを本カーソルと呼び、最終行にあるカーソルを表示選択カーソルと呼ぶ。最終行は表示選択行と呼ぶ。



《表示選択行》
表示選択行には、入力方式と入力文字の状態を次のように表示する。



《表示選択方式による入力》
表示選択方式で漢字を入力する場合には、直接入力方式のかなモード（ひらがなまたはカタカナ）か、ローマ字入力方式を使う。漢字１文字の読みを入力した後で【ＸＦＥＲ】を押すと、一度に最大九個の漢字一覧表を表示選択行に表示する。この操作を変換と呼ぶ。該当する漢字の番号を入力すると本カーソルの位置に漢字を表示する。該当する漢字がなければ再度【ＸＦＥＲ】を押す。漢字一覧表の最後まで進んだ後は一番始めの表に戻る。

入力した文字を漢字に変換せずそのまま入力する場合には、入力した後で【ＳＨＩＦＴ】と【ＸＦＥＲ】を押す。本カーソルの位置に文字を表示する。この操作を無変換と呼ぶ。

《ＪＩＳコード方式による入力》
ＪＩＳコード４桁（１６進）を入力すると漢字１文字を表示選択行に表示する。【ＳＨＩＦＴ】と【ＸＦＥＲ】を押すと本カーソルの位置に漢字を表示する。

《日本語入力モード時の特殊キーの働き》
日本語入力モードにある場合には、特殊キーの働きは標準の働きと異なる。

表示選択行に文字が入っていない場合には、カーソルや【ＩＮＳ】、【ＤＥＬ】キーは標準の働きをするが、表示選択行に文字が入っている場合には表示選択行に対してだけカーソルが有効になる。そして【→】と【←】は有効だが、【↑】、【↓】、【ＩＮＳ】、【ＤＥＬ】は無効になる。

日本語入力

【ＣＴＲＬ】＋【ＸＦＥＲ】	日本語入力モードの切り替え
【ＣＴＲＬ】＋【Ｖ】	直接入力、ローマ字入力の切り替え
【カナ】	英字モード、かなモードの切り替え
【ＣＴＲＬ】＋【Ｑ】	かなモードでひらがな、カタカナの切り替え
【ＣＡＰＳ】	英字モードで英大文字、英小文字の切り替え
【ＸＦＥＲ】	漢字へ変換
【ＳＨＩＦＴ】＋【ＸＦＥＲ】	表示選択行の文字を入力
【ＣＴＲＬ】＋【Ｔ】	全角文字、半角文字の切り替え
【ＣＴＲＬ】＋【Ｐ】	ＪＩＳコード入力方式にする

表示選択行の編集（表示選択行に文字がある場合）

【ＣＬＲ】	表示選択行に入力した内容をクリアーして、カーソルを先頭に戻す
【ＣＴＲＬ】＋【Ｕ】	
【ＣＴＲＬ】＋【Ｘ】	
【→】、【←】	表示選択行に入力した文字の範囲でカーソル移動
【ＨＯＭＥ】	表示選択行のカーソルを先頭に戻す
【ＣＴＲＬ】＋【Ｂ】	
【ＣＴＲＬ】＋【Ｆ】	
【ＣＴＲＬ】＋【Ｋ】	
【ＣＴＲＬ】＋【Ｎ】	
【ＣＴＲＬ】＋【Ｄ】	表示選択行のカーソル以後の文字を消す
【ＣＴＲＬ】＋【Ｅ】	
【ＲＥＴ】【↵】	日本語入力モードから抜け、１行の入力終了。表示選択行の内容は無視
【ＳＴＯＰ】	表示選択行の内容を無視して日本語入力モードから抜ける
ＥＳＣ、ＧＲＡＰＨ＋他のキー、Ｒｏｌｌ　ｕｐ、Ｒｏｌｌ　ｄｏｗｎ、↑、↓、ＩＮＳ、ＤＥＬ、ＢＳ、ＴＡＢ、ＣＴＲＬ＋Ｈ、ＣＴＲＬ＋Ｏ、ＣＴＲＬ＋Ｒ、ＣＴＲＬ＋Ｓはブザーが鳴って無視される。他のキーは標準と同じ	

固有の動作（表示選択行に文字がない場合）

【，】【ＲＥＴ】	日本語入力モードから抜け、１行の入力終了
ＳＴＯＰ	日本語入力モードから抜けＳＴＯＰ
ＥＳＣ、ＧＲＡＰＨ＋他のキー、Ｒｏｌｌ　ｕｐ、Ｒｏｌｌ　ｄｏｗｎ、ＣＴＲＬ＋Ｏはブザーが鳴って無視される。他のキーは標準と同じ	

ローマ字入力の規則

あ	い	う	え	お
a	i	u	e	o
か	き	く	け	こ
ka	ki	ku	ke	ko
さ	し	す	せ	そ
sa	si shi	su	se	so
た	ち	つ	て	と
ta	ti chi	tu tsu	te	to
な	に	ぬ	ね	の
na	ni	nu	ne	no
は	ひ	ふ	へ	ほ
ha	hi	hu fu	he	ho
ま	み	む	め	も
ma	mi	mu	me	me
や		ゆ		よ
ya		yu		yo
ら	り	る	れ	ろ
ra	ri	ru	re	ro
わ				を
wa				wo

長音記号　【＾】または【－】

促音　(1)子音を重ねる
(2)母音＋【　’　】

よう音　【ＳＨＩＦＴ】＋ローマ字

句読点　【　，　】

ん　(1)ｎの次に子音
(2)ｎ＋【　’　】

が	ぎ	ぐ	げ	ご
ga	gi	gu	ge	go
ざ	じ	ず	ぜ	ぞ
za	zi	zu	ze	zo
だ	ぢ	づ	で	ど
da	di	du	de	do
ば	び	ぶ	べ	ぼ
ba	bi	bu	be	bo
ぱ	ぴ	ぷ	ぺ	ぽ
pa	pi	pu	pe	po

きゃ	きゅ	きょ	しゃ	しゅ	しょ
kya	kyu	kyo	syasha	syushu	syosho
ちゃ	ちゅ	ちょ	にゃ	にゅ	にょ
tyacha	tyuchu	tyocho	nya	nyu	nyo
ひゃ	ひゅ	ひょ	みゃ	みゅ	みょ
hya	hyu	hyo	mya	myu	myo
りゃ	りゅ	りょ	ぎゃ	ぎゅ	ぎょ
rya	ryu	ryo	gya	gyu	gyo
じゃ	じゅ	じょ	ぢゃ	ぢゅ	ぢょ
zya ja	zyu ju	zyo jo	dya	dyu	dyo
びゃ	びゅ	びょ	ぴゃ	ぴゅ	ぴょ
bya	byu	byo	pya	pyu	pyo

カタカナの場合の特殊入力

ファ	フィ	フェ	フォ	ジェ	シェ
fa	fi	fe	fo	je	she
チェ	ヴァ	ヴィ	ヴ	ヴェ	ヴォ
che	va	vi	vu	ve	vo

12．プリンターの使い方

☆プリンターの操作☆

《基本事項》

（1）コンピューター本体とプリンターとが正しく接続されていること。
（2）電源が入っていること。
（3）オンライン状態であること。（セレクト（ＳＥＬ）ランプが点灯）
（4）用紙が正しくセットされていること。
（5）機種のちがい　・ＰＣ－８８２２……漢字ＲＯＭボードを内蔵（漢字はＪＩＳ第一水準）
　　　　　　　　　・ＰＣ－８８２１……漢字ＲＯＭボードはオプション
（6）プリンターの動作を制御する書式制御文字
　　ＣＲ（Carriage Return）：復帰……プリンターのヘッドを同一行の初めの文字位置にもどす
　　ＬＦ（Line Feed）：改行……プリンターのヘッド位置は変えず、次の行に進める
　　ＦＦ（Form Feed）：書式送り…プリンターのヘッド位置は変えず、次のページの頭（ＴＯＦ）に進める
　　ＶＴ（Vertical Tabulation）：垂直タブ…プリンターのヘッド位置は変えず、次のあらかじめ定められた行に進める

《ディップスイッチの設定》

標準的な使い方ではＳＷ１の６とＳＷ２の５だけＯＮにしておく。最初にプリンターにデータを送るときは、セレクト（ＳＥＬ）スイッチを押してオンライン状態にする必要がある。

番号		機能	ＯＮ	ＯＦＦ
ＳＷ１	1 2 3	使用する国別に文字の設定	日本ではすべてＯＦＦ	
	4	１／６インチ改行でのページ当たりの行数	７２行	６６行
	5	オンライン・オフライン指令コードの有効・無効	無効	有効
	6	バッファがいっぱいで印刷したときのＬＦの有・無	ＬＦ有	ＬＦ無
	7	ロジカルシークモードでの印字指令コードの設定	CR、LF、VT、FF	CRのみ
	8	ＣＲ指令コードを送ったときの動作を設定	CR＋LF	CRのみ
ＳＷ２	1	数字のゼロ表示の設定	φ	0
	2	未使用		
	3	１行当たりのドット数の設定	６４０ドット	９６０ドット
	4	未使用		
	5	印字方式の設定	高品質印字	高速印字
	6	データのビット数の設定	７ビット	８ビット
	7	電源投入時のオンライン・オフラインの設定	オンライン	オフライン
	8	印字方向の設定	単方向	双方向

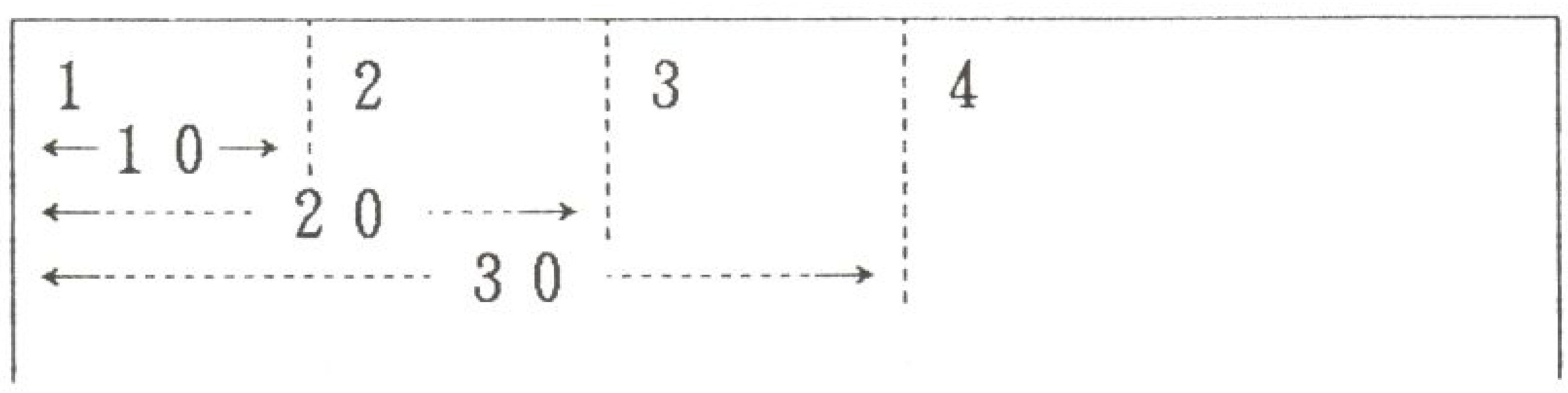
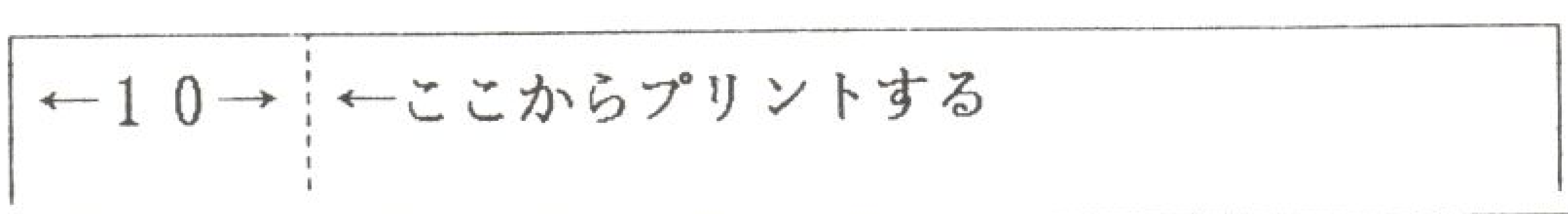
《基本操作》

命令	機能
ＬＦＩＬＥＳ	使い方は、ＦＩＬＥＳと同じ。プリンターへ出力する
ＬＬＩＳＴ	プログラムのリストをプリンターへ出力する
ＬＰＲＩＮＴ	データや計算結果をプリンターへ出力する
ＬＰＲＩＮＴ　ＵＳＩＮＧ	ＰＲＩＮＴ　ＵＳＩＮＧと同じ使い方で、プリンターへ出力する
ＬＰＲＩＮＴ　ＣＨＲ＄（１２）	改ページ
ＬＰＯＳ（ｘ）	ｘはダミー変数。プリンターヘッドの現在位置（横方向）を与える
ＷＩＤＴＨ　ＬＰＲＩＮＴ	プリンターの１行の文字数を設定する

☆プリンターの機能制御☆
《文字制御》

機　　　　能	命　　　　　　　　令
ＨＳ（高速印字）　パイカにする	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（７８）；
ＨＤ（高品質印字）パイカにする	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（７２）；
エリートにする	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（６９）；
コンデンスにする	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（８１）；
プロポーショナルにする	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（８０）；
漢字にする（漢字ＲＯＭが必要）	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（７５）；
英数記号とカタカナにする	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（３６）；
英数記号とひらがなにする	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（３８）；
拡大文字にする	ＬＰＲＩＮＴ　ＣＨＲ＄（１４）；
拡大文字の解除	ＬＰＲＩＮＴ　ＣＨＲ＄（１５）；
強調文字にする	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（３３）；
強調文字の解除	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（３４）；
アンダーラインを引く	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（８８）；
アンダーラインの解除	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（８９）；
文字を繰り返してプリントする	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（８２）； ”ｎ”；”ａ”　　ｎは繰り返す回数で３桁の１０進数 ａは繰り返してプリントする文字
１列８ドットのドット列印字モードにする	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（８３）； ”ｎ”；　　ｎは送るドット列数で４桁の１０進数
１列１６ドットのドット列印字モードにする	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（７３）； ”ｎ”；　　ｎは送るドット列数で４桁の１０進数
１列８ドットのパターンを繰り返しプリントする	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（８６）； ”ｎ”；”ａ”　　ｎは繰り返す回数で４桁の１０進数 ａは繰り返しプリントするドット列
１列１６ドットのパターンを繰り返しプリントする	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（８７）； ”ｎ”；”ａ”　　ｎは繰り返す回数で４桁の１０進数 ａは繰り返しプリントするドット列
外字を作る（漢字ＲＯＭが必要）	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（４２）； ＬＰＲＩＮＴ　ＣＨＲ＄（＆Ｈｘｘ）；ＣＨＲ＄（＆Ｈｙｙ）； ＬＰＲＩＮＴ　ＣＨＲ＄（４）； 外字の漢字コードとして、７６２０Ｈ～７６５ＦＨの６４文字分が定義できる。定義する漢字コードの上２桁をｘｘ、下２桁をｙｙに書く　（例５参照）

《幅と行と位置の制御》

機　能	命　令
改行幅を1／6㌅にする	LPRINT CHR$(27);CHR$(65); または LPRINT CHR$(27);"A";でも同じ
改行幅を1／8㌅にする	LPRINT CHR$(27);CHR$(66);
n／120㌅の改行幅にする	LPRINT CHR$(27);CHR$(84);"n"; 00=<n=<99のこと。nは2桁の10進数
水平タブを設定する	LPRINT CHR$(27);CHR$(40);"a,b,c,··p."; a～pは3桁の10進数。10は010と書くこと。コンマ(,)は継続コード、ピリオド(.)は終了コード。a～pは16個まで設定できる (例) 100 LPRINT CHR$(27);CHR$(40); "010,020,030."; 110 FOR I=1 TO 4 120 LPRINT I; 130 GOSUB 500 140 NEXT I 150 END 実行結果  プリントする場合、以下のサブルーチンを実行し、水平タブコードをプリンターへ送っておくこと。水平タブを解除するときにも必要 500 S=INP(&H42) AND 1:IF S THEN 500 510 OUT &H40,9:OUT &H44,&H8: OUT &H44,&H88 520 RETURN (88)のとき500と510行を変える 500 S=INP(&H40) AND 1:IF S THEN 500 510 OUT &H10,9:OUT &H43,0:OUT &H43,1
水平タブを全部解除する	LPRINT CHR$(27);CHR$(50);
水平タブを部分解除する	LPRINT CHR$(27);CHR$(41);"a,b."; (例) 前の例のうち、10と20の水平タブを解除する LPRINT CHR$(27);CHR$(41);"010,020";
レフトマージンを設定する	LPRINT CHR$(27);CHR$(76);"n"; nは3桁の10進数で記入する (例) LPRINT CHR$(27);CHR$(76);"010"; 
レフトマージンを解除する	LPRINT CHR$(27);CHR$(76);"000";

機　能	命　令
垂直タブの設定	電源投入時のページの頭（ＴＯＦ）と垂直タブ（ＶＴ）

行	設定
1	ＴＯＦ（ｃｈ１）
7	ＶＴ　（ｃｈ２）
13	ＶＴ　（ｃｈ２）
55	ＶＴ　（ｃｈ２）
61	ＶＴ　（ｃｈ２）
66	
67	ＴＯＦ（ｃｈ１）

・ＴＯＦから次のＴＯＦの前までが１ページあたりの行数（６６行）

・ＶＴはｃｈ２に６行毎に初期設定される

・ＬＰＲＩＮＴ　ＣＨＲ＄（１１）；で、次のＶＴまで改行する

・ＶＴを設定しなおせば、自由に改行できる

・ＶＴはｃｈ２～ｃｈ６に５種類設定できる

・ＴＯＦはｃｈ１に設定する

１頁あたり６０行とし、ｃｈ３にＶＴを１０行毎に設定する

行	設定
1	ＴＯＦ（ｃｈ１）
10	ＶＴ　（ｃｈ３）
20	ＶＴ　（ｃｈ３）
50	ＶＴ　（ｃｈ３）
60	ＶＴ　（ｃｈ３）
61	ＴＯＦ（ｃｈ１）

・２バイトでＴＯＦまたはＶＴを設定する。
１行目と６１行目は、ｃｈ１にＴＯＦを設定し、ｃｈ３に１０行毎にＶＴを設定する

データの意味

１バイト目	X	1	ch6	ch5	ch4	ch3	ch2	ch1
２バイト目	X	1	X	X	X	X	X	X

X　：０でも１でも可（この例では０）

chx　：ＴＯＦを設定する行のデータの場合、ch1のビットを１にする
ＶＴを設定する行のデータの場合、設定するchx のビットを１にする

```
 10 DATA 65,64,64,64,64,64,64,64,64,68
 20 DATA 64,64,64,64,64,64,64,64,64,68
 30 DATA 64,64,64,64,64,64,64,64,64,68
 40 DATA 64,64,64,64,64,64,64,64,64,68
 50 DATA 64,64,64,64,64,64,64,64,64,68
 60 DATA 64,64,64,64,64,64,64,64,64,68
 70 DATA 65
 80 LPRINT CHR$(29);
 90 FOR K=1 TO 61
100   READ D:LPRINT CHR$(D);CHR$(64);
110 NEXT K
120 LPRINT CHR$(30);
130 LPRINT STRING$(10,"*")
140 LPRINT CHR$(31);CHR$(3);
150 LPRINT STRING$(10,"@")
160 LPRINT CHR$(31);CHR$(3);
170 LPRINT STRING$(10,"$")
```

８０行目……垂直タブ設定開始宣言
１００行目……ＴＯＦ、ＶＴの設定
１２０行目……垂直タブ設定終了宣言
１４０、１６０行目……ｃｈ３に設定した垂直タブを実行

プログラムを実行すると、１行目、１０行目、２０行目に＊、＠、＄をそれぞれ１０個ずつプリントする

機　　能	命　　　　令
垂直タブ	ＬＰＲＩＮＴ　ＣＨＲ＄（１１）； ch2 に設定した次の垂直タブまで改行する。垂直タブを設定してないときは、次のＴＯＦまで改行する。垂直タブはch2 に6行毎に初期設定されている
1行改行する	ＬＰＲＩＮＴ　ＣＨＲ＄（１０）；
n行改行する	ＬＰＲＩＮＴ　ＣＨＲ＄（３１）；ＣＨＲ＄（１６＋n）； ただし、１＝＜ｎ＝＜１５
ドット単位でプリント開始位置を一時的に設定	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（７０）；”ｎ”； ｎは4桁の１０進数で記入する。ただし、０＝＜ｎ＝＜９５９ (例) レフトマージンから２５ドットあけてプリントする １００　ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（７０）；”００２５”； １１０　ＬＰＲＩＮＴ　”ＡＢＣＤＥ”；
ドットスペースをプリントする	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（ｎ）； ただし、１＝＜ｎ＝＜６ プロポーショナルまたは漢字のときのみ１～６ドットのスペースがとれる

《その他の制御》

機　　能	命　　　　令
1行にプリントするドット数を６４０にする	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（６８）； 1列８ドットのドット列印字モードのときに有効
1行にプリントするドット数を９６０にする	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（７７）； 電源投入時にはこの値がセットされる。コンデンスモードのときは１９２０になる
ロジカルシークモードにする	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（６０）； 印字終了時点から次の印字指令などでプリンターヘッドが距離の近い方に動く
片方向印字モードにする	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（６２）；
改行を順方向にする	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（１０２）
改行を逆方向にする	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（１１４）
プリンターリボンの切り換え	ＬＰＲＩＮＴ　ＣＨＲ＄（２７）；ＣＨＲ＄（６７）；”ｎ”； ｎ＝０のとき　黒リボン ｎ＝２のとき　赤リボン
プリンターをオンラインにする	ＯＵＴ　＆Ｈ４０，＆Ｈ１１：ＯＵＴ　＆Ｈ４４，＆Ｈ８： ＯＵＴ　＆Ｈ４４，＆Ｈ８８ （８８） ＯＵＴ　＆Ｈ１０，＆Ｈ１１：ＯＵＴ　＆Ｈ４３，０：ＯＵＴ　＆Ｈ４３，１
プリンターをオフラインにする	ＯＵＴ　＆Ｈ４０，＆Ｈ１３：ＯＵＴ　＆Ｈ４４，＆Ｈ８： ＯＵＴ　＆Ｈ４４，＆Ｈ８８ （８８） ＯＵＴ　＆Ｈ１０，＆Ｈ１３：ＯＵＴ　＆Ｈ４３，０：ＯＵＴ　＆Ｈ４３，１

機　　能	命　　　　令
一番近い水平タブにプリンターのヘッドを移動	10 S=INP(&H42) AND 1:IF S THEN 10 20 OUT &H40,9:OUT &H44,&H8: OUT &H44,&H88 (88) 10 S=INP(&H40) AND 1:IF S THEN 10 20 OUT &H10,9:OUT &H43,0:OUT &H43,1

《印字見本》

（1）HSパイカ文字

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMN
OPQRSTUVWXYZ[¥]^_`abcdefghijklmnopqrstuvwxyz{¦
}~ ▁▂▃▄▅▆▇█▏▎▍▌▋▊▉┼┴┬┤├▔▕┌┐└┘╭╮╰╯ ｡｢｣､･ｦｧｨｩｪ
ｫｬｭｮｯｰｱｲｳｴｵｶｷｸｹｺｻｼｽｾｿﾀﾁﾂﾃﾄﾅﾆﾇﾈﾉﾊﾋﾌﾍﾎﾏﾐﾑﾒﾓﾔﾕﾖﾗﾘ
ﾙﾚﾛﾜﾝﾞﾟ═╞╪╡◢◣◥◤♠♥♦♣●○╱╲╳円年月日時分秒
```

（2）HDパイカ文字

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMN
OPQRSTUVWXYZ[¥]^_`abcdefghijklmnopqrstuvwxyz{¦
}~ ▁▂▃▄▅▆▇█▏▎▍▌▋▊▉┼┴┬┤├▔▕┌┐└┘╭╮╰╯ ｡｢｣､･ｦｧｨｩｪ
ｫｬｭｮｯｰｱｲｳｴｵｶｷｸｹｺｻｼｽｾｿﾀﾁﾂﾃﾄﾅﾆﾇﾈﾉﾊﾋﾌﾍﾎﾏﾐﾑﾒﾓﾔﾕﾖﾗﾘ
ﾙﾚﾛﾜﾝﾞﾟ═╞╪╡◢◣◥◤♠♥♦♣●○╱╲╳円年月日時分秒
```

（3）エリート文字

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMN
OPQRSTUVWXYZ[¥]^_`abcdefghijklmnopqrstuvwxyz{¦
}~ ▁▂▃▄▅▆▇█▏▎▍▌▋▊▉┼┴┬┤├▔▕┌┐└┘╭╮╰╯ ｡｢｣､･ｦｧｨｩｪ
ｫｬｭｮｯｰｱｲｳｴｵｶｷｸｹｺｻｼｽｾｿﾀﾁﾂﾃﾄﾅﾆﾇﾈﾉﾊﾋﾌﾍﾎﾏﾐﾑﾒﾓﾔﾕﾖﾗﾘ
ﾙﾚﾛﾜﾝﾞﾟ═╞╪╡◢◣◥◤♠♥♦♣●○╱╲╳円年月日時分秒
```

（4）コンデンス文字

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMN
OPQRSTUVWXYZ[¥]^_`abcdefghijklmnopqrstuvwxyz{¦
}~ ▁▂▃▄▅▆▇█▏▎▍▌▋▊▉┼┴┬┤├▔▕┌┐└┘╭╮╰╯ ｡｢｣､･ｦｧｨｩｪ
ｫｬｭｮｯｰｱｲｳｴｵｶｷｸｹｺｻｼｽｾｿﾀﾁﾂﾃﾄﾅﾆﾇﾈﾉﾊﾋﾌﾍﾎﾏﾐﾑﾒﾓﾔﾕﾖﾗﾘ
ﾙﾚﾛﾜﾝﾞﾟ═╞╪╡◢◣◥◤♠♥♦♣●○╱╲╳円年月日時分秒
```

（5）プロポーショナル文字

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMN
OPQRSTUVWXYZ[¥]^_`abcdefghijklmnopqrstuvwxyz{¦
}~ ▁▂▃▄▅▆▇█▏▎▍▌▋▊▉┼┴┬┤├▔▕┌┐└┘╭╮╰╯ ｡｢｣､･ｦｧｨｩｪ
ｫｬｭｮｯｰｱｲｳｴｵｶｷｸｹｺｻｼｽｾｿﾀﾁﾂﾃﾄﾅﾆﾇﾈﾉﾊﾋﾌﾍﾎﾏﾐﾑﾒﾓﾔﾕﾖﾗﾘ
ﾙﾚﾛﾜﾝﾞﾟ═╞╪╡◢◣◥◤♠♥♦♣●○╱╲╳円年月日時分秒
```

（6）漢字

亜唖娃阿哀愛挨姶逢葵茜穐悪握渥旭葦芦鯵梓圧斡扱
宛姐虻飴絢綾鮎或粟袷安庵按暗案闇鞍杏以伊位依偉
囲夷委威尉惟意慰易椅為畏異移維緯胃萎衣謂違遺医

《ドットのパターン出力例》

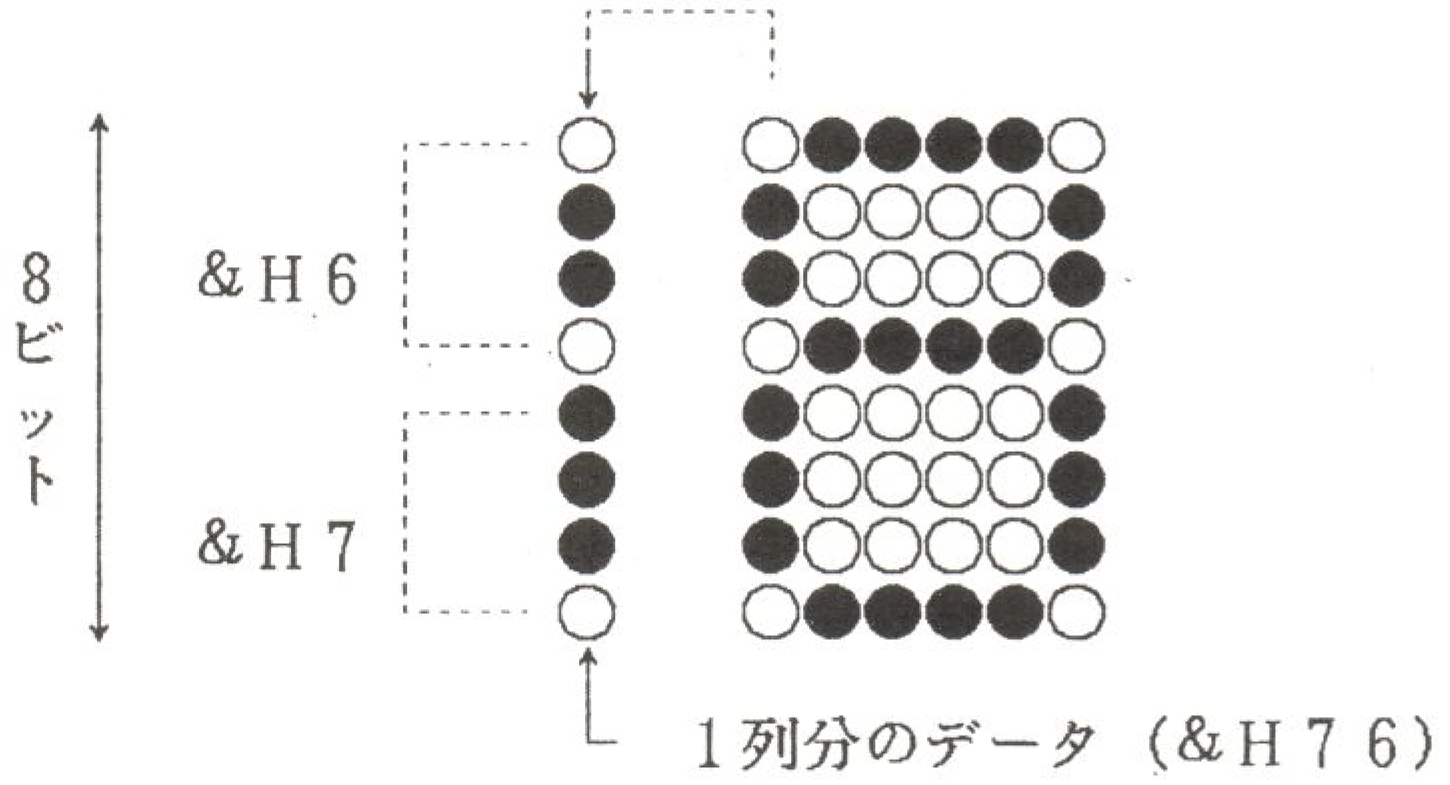
（例1）1列8ドットのパターンをプリントする。

```
  10 LPRINT CHR$(27);CHR$(83);"0006";
  20 LPRINT CHR$(&H76);
  30 LPRINT STRING$(4,&H89);
  40 LPRINT CHR$(&H76);
```

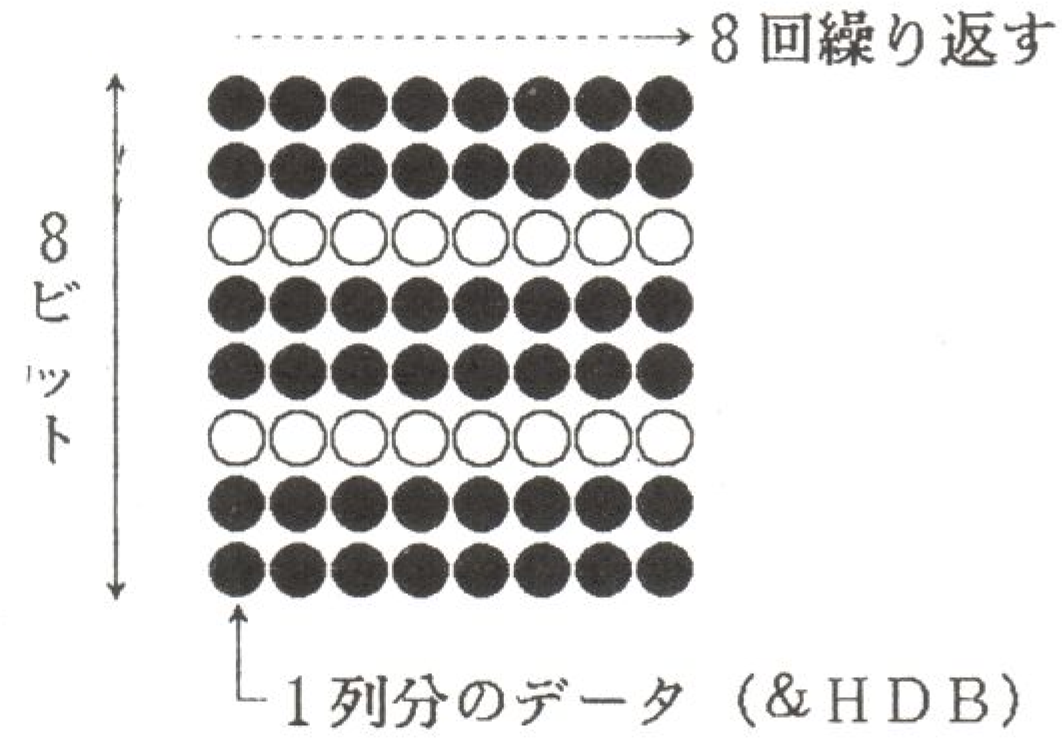
（例2）1列8ドットのパターンを繰り返しプリントする。

```
  10 LPRINT CHR$(27);CHR$(86);"0008";CHR$(&HDB);
```

例1の結果：（ドット列＝０００６）



例2の結果：（繰り返し＝０００８）



（例3）1列16ドットのパターンをプリントする。

```
  10 LPRINT CHR$(27);CHR$(73);"0016";
  20 FOR I=1 TO 32
  30 READ A$ : C=VAL("&H"+A$) : GOSUB 500
  40 NEXT I : LPRINT
 100 DATA 00,00,04,10,F4,17,94,14,9F,FC
 110 DATA 94,14,F4,17,04,10,00,E0,FE,1F
 120 DATA 22,02,22,02,22,82,22,82,FE,FF,00,00
 130 END
 500 S=INP(&H42) AND 1 : IF S THEN 500
 510 OUT &H40,C : OUT &H44,&H8 : OUT &H44,&H88
 520 RETURN
```

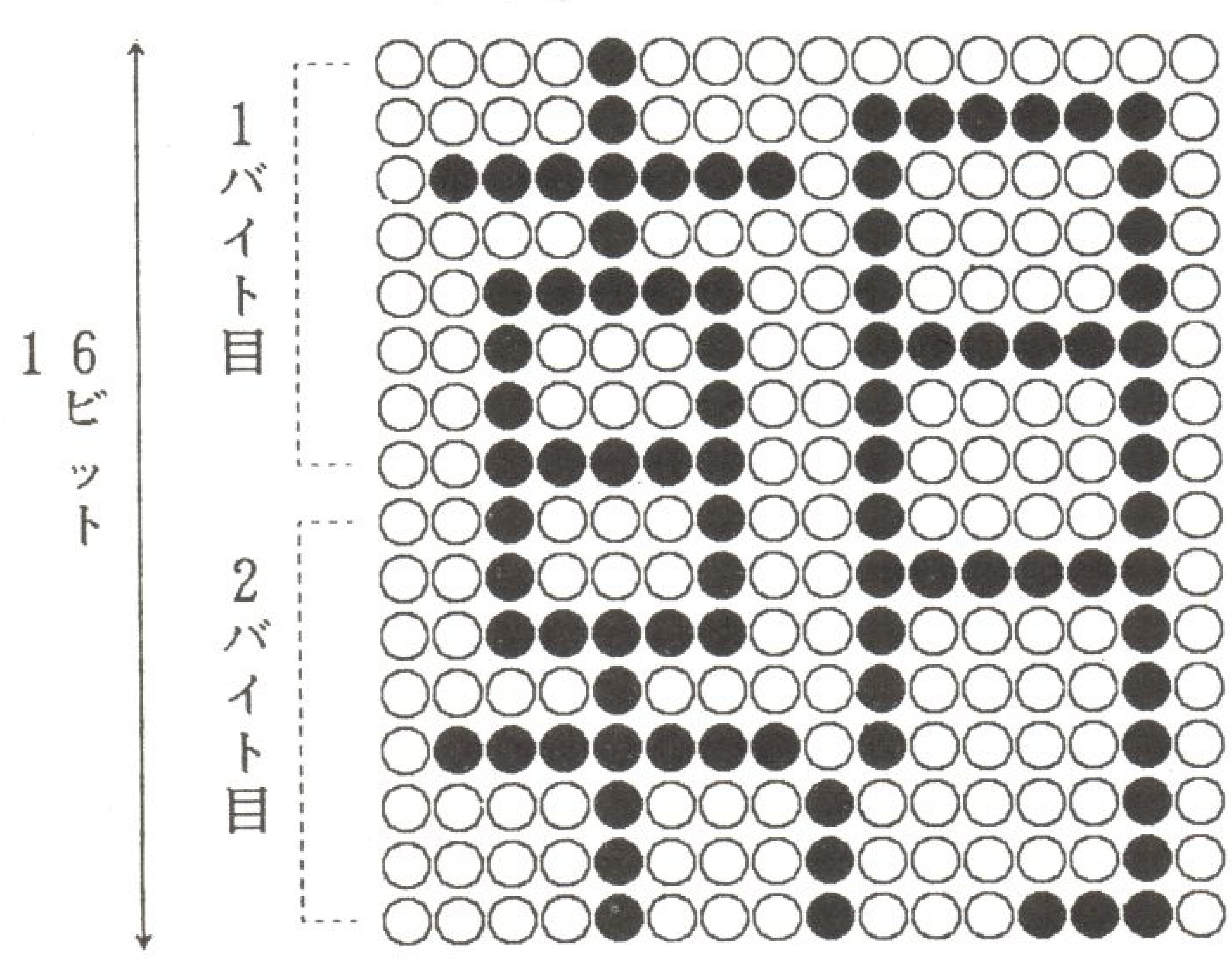
※（88）のときは、行番号500と510を次のようにする。

```
 500 S=INP(&H40) AND 1 : IF S THEN 500
 510 OUT &H10,C : OUT &H43,0 : OUT &H43,1
```

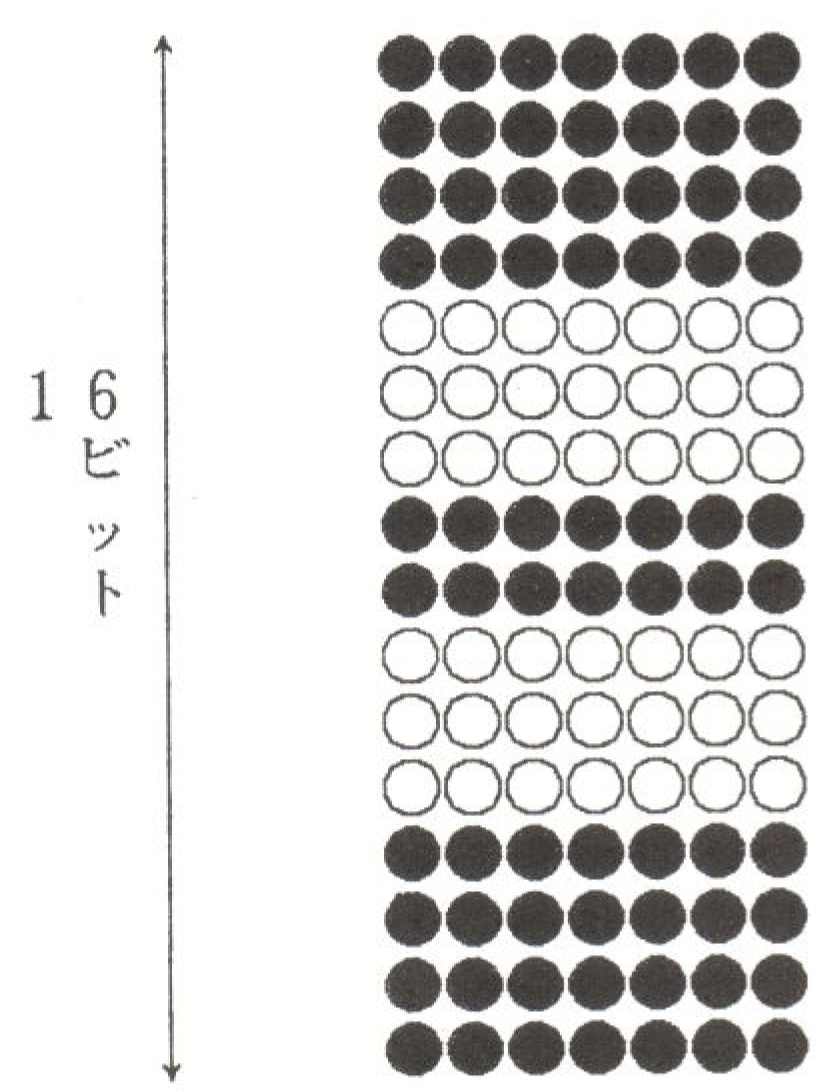
（例4）1列16ドットのパターンを繰り返しプリントする。

```
  10 LPRINT CHR$(27);CHR$(87);"0007";CHR$(&H8F)
      ;CHR$(&HF1);
```

例3の結果：（ドット列＝００１６）



例4の結果：（繰り返し＝０００７）



（例５）"①"という外字を、漢字コード７６２０Ｈと定義して作る。

```
 10 LPRINT CHR$(27);CHR$(42);
 20 LPRINT CHR$(&H76);CHR$(&H20);
 30 FOR I=1 TO 32
 40 READ A$ : D=VAL("&H"+A$) : GOSUB 500
 50 NEXT I
 60 LPRINT CHR$(4);
 70 END
200 DATA E0,07,18,18,04,20,04,20
210 DATA 02,40,02,40,11,90,F9,9F
220 DATA F9,9F,01,90,02,40,02,40
230 DATA 04,20,04,20,18,18,E0,07
500 S=INP(&H42) AND 1 : IF S THEN 500
510 OUT &H40,D : OUT &H44,&H8 : OUT &H44,&H88
520 RETURN
```

（８８）のときは、行番号５００と５１０行を次のように変える。

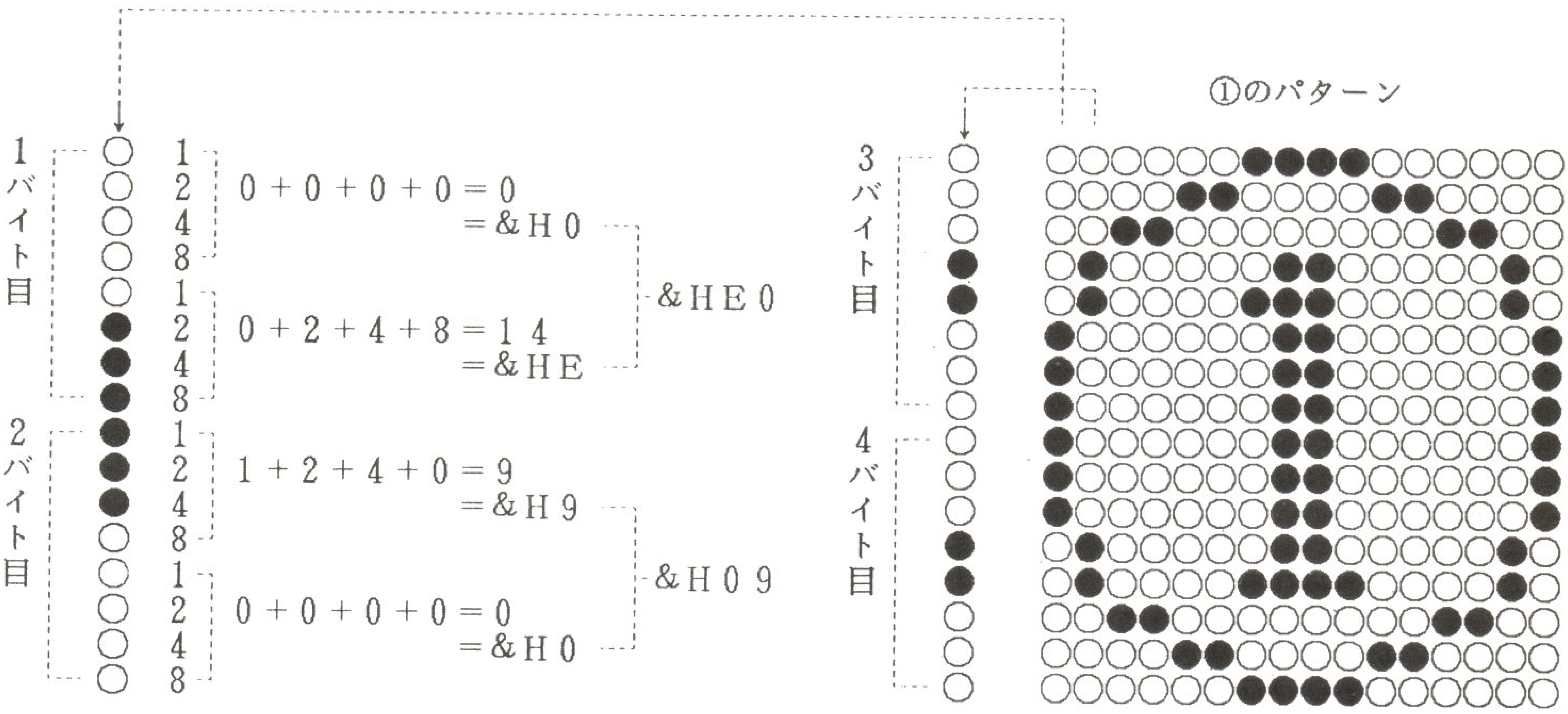
```
500 S=INP(&H40) AND 1 : IF S THEN 500
510 OUT &H10,D : OUT &H43,0 : OUT &H43,1
```

このプログラムを実行すると"①"という外字ができる。

プリントするには、以下の行を追加する。

```
65 LPRINT CHR$(27);CHR$(75);
66 LPRINT CHR$(&H76);CHR$(&H20);
67 LPRINT CHR$(27);CHR$(72):LPRINT
```

外字を作るには、１６×１６の升目を準備し外字のパターンを作る。パターンができたら下記のように１バイト単位に分解して、データを作る。



●はドットをうつ
○はドットをうたない

13．フロッピーディスクの使い方

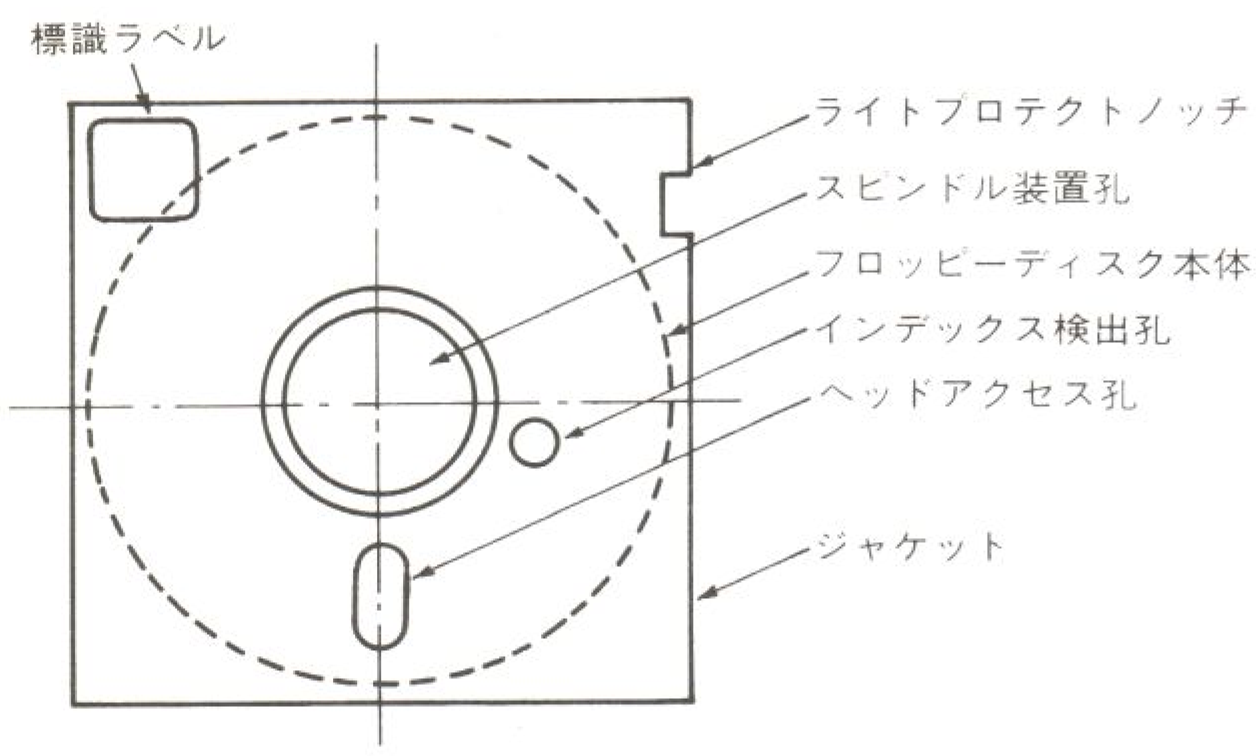
☆基本事項☆

《フロッピーディスク装置の種類》

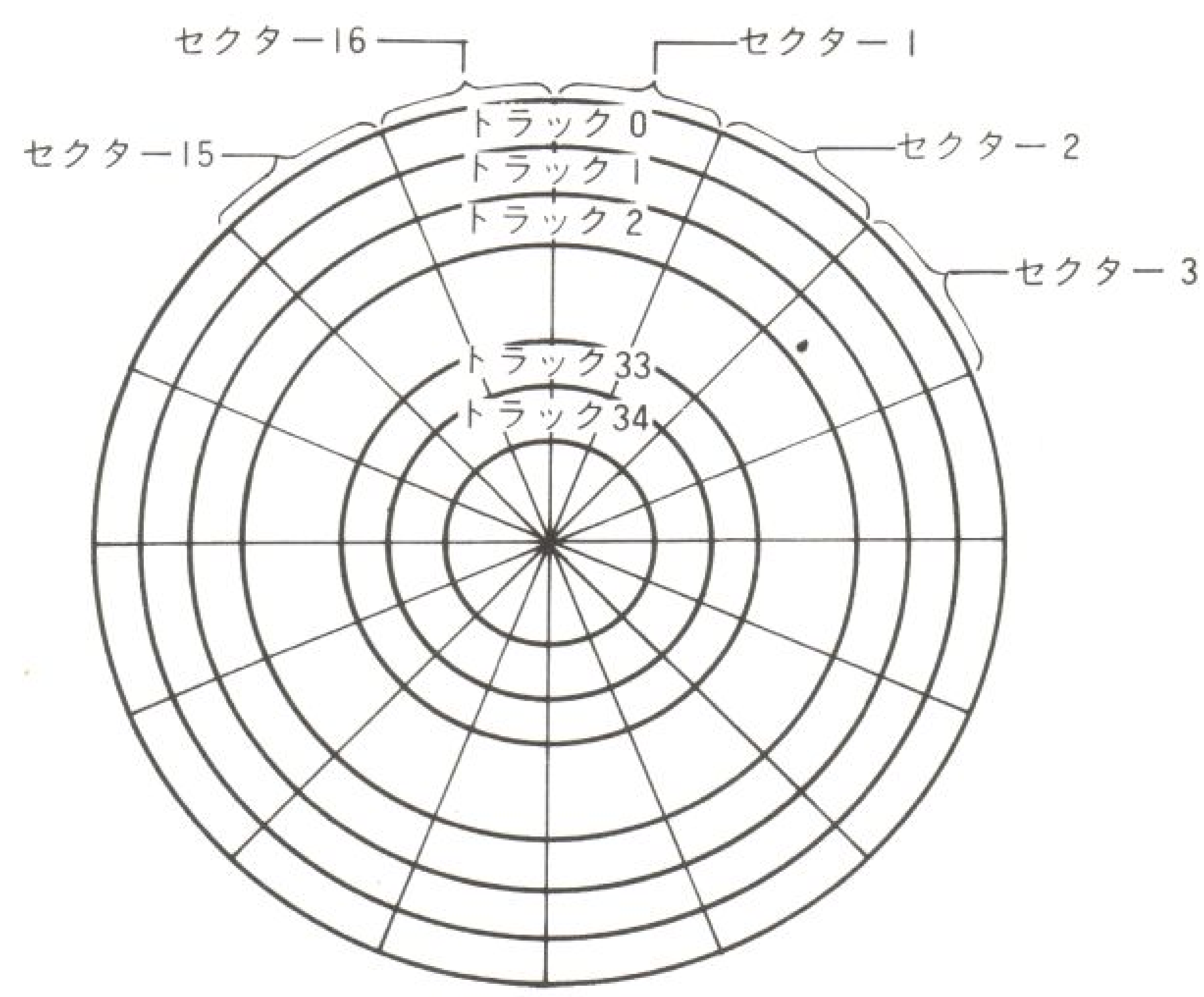
PC－9881　※	8インチ標準フロッピーディスク（両面倍密度用）	1MB×2ドライブ
PC－8881	8インチ標準フロッピーディスク（両面倍密度用）	1MB×2ドライブ
PC－80S31	5インチミニフロッピーディスク（両面倍密度用）	320KB×2ドライブ
PC－8031－1W	5インチミニフロッピーディスク（片面倍密度用）	143KB×2ドライブ
PC－8031－1V	5インチミニフロッピーディスク（片面倍密度用）	143KB×1ドライブ

※　PC－9801でのみ使用可能。

《ミニフロッピーディスクの外形》



《ミニフロッピーディスクの構造（片面倍密）》



《フロッピーディスク装置とドライブ番号》

各フロッピーディスク装置に割り振られるドライブ番号は、フロッピー装置の種類や組み合わせによって異なる。主な組み合わせは次のとおり。

接続するフロッピー装置		割り振られるドライブ番号	
PC－9881 または PC－8881	PC－8031－1W または PC－80S31	PC－9881 または PC－8881	PC－8031－1W または PC－80S31
○	×	1～2	－
×	○	－	1～2
○	○	1～2	3～4

○はフロッピー装置の接続を示す

《ＤＩＳＫ－ＢＡＳＩＣのスタート》

（１）フロッピーディスク装置の電源を入れる。

（２）システムディスクを装着する。

（３）ＰＣ本体の電源を入れる。
　ＰＣ－９８８１ディスク装置の電源は、ＰＣ－９８０１の電源を入れると入る。したがってＰＣ－９８０１の電源を入れてからシステムディスクを装着する。

（４）システムプログラムが読み込まれる。

（５）画面に次の表示が出る。（８８）とあるのは、ＰＣ－８８０１のときに表示するものを示す。

```
Ｄｉｓｋ　ｖｅｒｓｉｏｎ
Ｄｉｓｋ　ｖｅｒｓｉｏｎ　〔Ａｕｇ　２０，１９８２〕 ………→（８８）
Ｈｏｗ　ｍａｎｙ　ｆｉｌｅｓ（０－１５）？
```

（６）同時にオ－プンできるファイルの数は１５までなので、１５以下の数字を入力する。

（７）画面に次の表示が出て準備完了。

```
ＮＥＣ　Ｎ－８８　ＢＡＳＩＣ（８６）ｖｅｒｓｉｏｎ　１．０
ＮＥＣ　Ｎ－８８　ＢＡＳＩＣ　Ｖｅｒｓｉｏｎ　１．１ ………→（８８）
Ｃｏｐｙｒｉｇｈｔ　（Ｃ）　１９８１　ｂｙ　Ｍｉｃｒｏｓｏｆｔ ……→（８８）
ｘｘｘｘｘｘ　Ｂｙｔｅｓ　ｆｒｅｅ
ＯＫ
```

ｘｘｘｘｘｘ：6桁の数字、（８８）は5桁。使えるメモリーの大きさを表す

《システムディスクの装着》

ＤＩＳＫ－ＢＡＳＩＣのスタート時にシステムディスクを装着するドライブ番号は、フロッピーディスク装置の種類や数によって変わる。

・８インチ標準フロッピーディスク装置だけのとき。
　どのドライブに装着してもよい。ドライブ１から順にシステムディスクをさがす。

・８インチ標準フロッピーディスク装置とミニフロッピーディスク装置のとき。
　８インチ標準フロッピーディスク装置ならどのドライブに装着してもよい。ミニフロッピーディスク装置ならドライブ１に装着する。

・ミニフロッピーディスク装置だけのとき。
　必ずドライブ１に装着する。

《システムディスク》

フロッピーディスクを使えるＢＡＳＩＣをＤＩＳＫ－ＢＡＳＩＣという。このＤＩＳＫ－ＢＡＳＩＣの入っているフロッピーディスクをシステムディスクといい、ＤＩＳＫ－ＢＡＳＩＣのスタート時に使う。一度ＤＩＳＫ－ＢＡＳＩＣが動き出せば、システムディスクをはずしてもよい。

システムディスクは、ＤＩＳＫ－ＢＡＳＩＣが入っている分だけ自由に使える領域が少ない。データやプログラムを保存するだけであれば、ＤＩＳＫ－ＢＡＳＩＣが入っていないフロッピーディスク（データディスクという）を使ったほうが、使える領域が多い。

システムディスクにはＤＩＳＫ－ＢＡＳＩＣのほかに、フォーマットプログラムなどの各種ユーティリティープログラムが入っている。

《バッファの個数》

ＢＡＳＩＣスタート時に表示する「Ｈｏｗ　ｍａｎｙ　ｆｉｌｅｓ（０－１５）？」に対しては、同時にオープンするファイル数を指定する。ＲＥＴＵＲＮキーだけ入力するとファイル数は以下のようになる。

・フロッピーディスクがつながっていないときは2　（８８）は1

・フロッピーディスクがつながっているときにはドライブ数

フロッピーディスクとの入出力に使うバッファは、指定したファイルの数だけ用意する。どのバッファを使うかは、ＯＰＥＮで指定するファイル番号によって決まる。

（例）　Ｈｏｗ　ｍａｎｙ　ｆｉｌｅｓ（０－１５）？に２と答えたとき

```
                                          用意するバッファ
                                       ┌──────────────┐
                                       │ ＃０のバッファ │
                                       ├──────────────┤
ＯＰＥＮ　”ＴＥＳＴ１”　ＡＳ　＃１………→│ ＃１のバッファ │
                                       ├──────────────┤
ＯＰＥＮ　”ＴＥＳＴ２”　ＡＳ　＃２………→│ ＃２のバッファ │
                                       └──────────────┘
ＯＰＥＮ　”ＴＥＳＴ３”　ＡＳ　＃３とはできない
```

《フォーマットの方法》
フロッピーディスクの入出力ができるようにする作業をディスクのフォーマッティングという。新しいフロッピーディスクを使うときには、必ずフォーマッティングが必要。
フォーマッティングには以下のような２つのレベルがある。

実行方法	レベル	内　容
システムディスクのｆｏｒｍａｔ．ｎ８８プログラムを使う	１	ＢＡＳＩＣのＤＳＫＩ＄・ＤＳＫＯ＄でのみ、読み書きできるようになる。ＦＡＴ、ディレクトリー、ＩＤの初期化はされないので通常のファイルを作ることはできない。市販の８インチ標準のフロッピーディスクは、この作業をしてあるので必要ない
	２	ＦＡＴ、ディレクトリー、ＩＤの初期化をする ＢＡＳＩＣのコマンド、ステートメントを使って、ファイルを作ることができる

〔ｆｏｒｍａｔ．ｎ８８プログラムの実行〕
このプログラムを使うと、フロッピーディスクのフォーマットのほかに、システムディスクも作れる。

（１）システムディスクからｆｏｒｍａｔ．ｎ８８プログラムをロードし、実行する。

（２）

```
Mount a new disk on even drive#
Format drive#?
```

偶数番号のドライブに新しいディスクを装着し、ドライブ番号【ＲＥＴ】。

（３）

```
Do you need physical formatting(y/n)?
```

ミニフロッピーディスクのときだけ表示。レベル１のフォーマッティングをする必要があれば、ｙ【ＲＥＴ】、なければ、ｎ【ＲＥＴ】。

ｙ【ＲＥＴ】のとき、

```
Formatting a disk ドライブ番号
```

を表示し、レベル１のフォーマッティングを開始する。

（４）

```
Create system disk(y/n)?
```

システムディスクを作るなら、ｙ【ＲＥＴ】。データディスクを作るなら、ｎ【ＲＥＴ】。

```
working
```

を表示して、レベル２のフォーマッティングを開始する。

（５）

```
System disk on drive#?
```

システムディスクを作るときだけ表示。システムディスクを装着し、ドライブ番号【ＲＥＴ】。

```
Sure(y/n)?
```

確認して正しければ、ｙ【ＲＥＴ】。まちがっていれば、ｎ【ＲＥＴ】で（５）にもどる。

```
Copying system
```

を表示し、ＤＩＳＫ－ＢＡＳＩＣをコピーする。

（６）

```
Completed.
```

を表示し、終了。

《バックアップファイルの取り方》
ディスクに記憶されているファイルは、使用中のミスによりこわれてしまうことがある。重要なファイルが記憶されているデータディスクやシステムディスクなどは、あらかじめ複写しておいたほうがよい。複写するには、システムディスクに入っているｂａｃｋｕｐ．ｎ８８プログラムを使う。プログラムを実行すると複写元のフロッピーの内容はＤＩＳＫ－ＢＡＳＩＣも含め、すべて複写先のフロッピーにコピーする。

〔ｂａｃｋｕｐ．ｎ８８プログラムの実行〕

（１）システムディスクからｂａｃｋｕｐ．ｎ８８プログラムをロードし、実行する。

（２）
```
Ｂａｃｋ　ｕｐ　ａ　ｄｉｓｋ
Ｍｏｕｎｔ　ｍａｓｔｅｒ　ｄｉｓｋ　ｏｎ　ｄｒｉｖｅ
Ｍａｓｔｅｒ　ｄｒｉｖｅ＃？
```
複写元のフロッピーをドライブに装着し、ドライブ番号【ＲＥＴ】。

（３）
```
Ｍｏｕｎｔ　ｎｅｗ　ｄｉｓｋ　ｏｎ　ｄｒｉｖｅ
Ｎｅｗ　ｄｒｉｖｅ＃？
```
複写先のフロッピーをドライブに装着し、ドライブ番号【ＲＥＴ】。

```
Ｂａｄ　ｄｒｉｖｅ＃
```
マスターファイルとバックアップファイルのドライブ番号が等しいとき表示し、（３）へもどる。

```
Ｃａｎ’ｔ　ｂａｃｋ　ｕｐ．
```
フロッピーディスクの種類が異なるか、ドライブ番号の入力がまちがっているとき表示する。（１）からやりなおす。

（４）８インチフロッピーディスクのときは（５）へ。
```
Ｄｏ　ｙｏｕ　ｎｅｅｄ　ｐｈｙｓｉｃａｌ　ｆｏｒｍａｔｔｉｎｇ（ｙ／ｎ）？
```
ミニフロッピーディスクのときだけ表示。レベル１のフォーマッティングが必要ならば、ｙ【ＲＥＴ】。必要なければ、ｎ【ＲＥＴ】。

ｙ【ＲＥＴ】のとき
```
Ｐｈｙｓｉｃａｌ　ｆｏｒｍａｔ　ａ　ｄｉｓｋ　ドライブ番号
Ｓｕｒｅ（ｙ／ｎ）？
```
確認して正しければ、ｙ【ＲＥＴ】。まちがっていれば、ｎ【ＲＥＴ】で(2)へもどる。

```
Ｆｏｒｍａｔｔｉｎｇ　ａ　ｄｉｓｋ　ドライブ番号
```
を表示し、レベル１のフォーマッティングをする。

（５）
```
Ｓｕｒｅ（ｙ／ｎ）？
```
標準フロッピーディスクのとき表示。確認して正しければ、ｙ【ＲＥＴ】。まちがっていれば、ｎ【ＲＥＴ】で（２）へもどる。

```
Ｃｏｐｙｉｎｇ　ｔｒａｃｋ　トラック番号
```
を表示し、マスターファイルをバックアップファイルにコピーする。

（６）
```
Ｃｏｍｐｌｅｔｅｄ．
```
を表示し終了。

☆データファイルの入出力☆

《ファイルとは》
項目（アイテム）の集まりをレコードといい、レコードの集まりをファイルという。

社員コード	名前	売り上げ	社員コード	名前	売り上げ	………………	社員コード	名前	売り上げ

項目
レコード
ファイル

《ファイルの種類》
ＢＡＳＩＣで扱うファイルには、プログラムファイルとデータファイルがある。プログラムファイルとはプログラムをＳＡＶＥ命令などで保存したものをいう。データファイルとは、プログラムで読み書きするデータを入れるファイルをさす。データファイルには、次の２種類ある。

・シーケンシャルファイル
「順編成ファイル」とも呼ぶ。レコードを順番にならべていったファイル。１０番目にあるレコードだけを読みたいときでも、１番目のレコードから順番に読んでいかなければならない。また、特定のレコードを削除・変更するときは、読み込んだファイルをもとに削除・変更し、その結果を新しいファイルに書き出さなくてはならない。レコードの長さは、一定でなくてもよい。

・ランダムファイル
「乱編成ファイル」とも呼ぶ。レコード番号を指定すれば、どのレコードでも、直接読み書きできる。レコードの長さは、２５６バイトで一定。
ランダムファイルを使うときは、ＦＩＥＬＤ文でバッファの項目を定義する。ＰＵＴ命令でバッファからディスクへ書き出し、ＧＥＴ命令でディスクからバッファへ読み込む。

《ＯＰＥＮ命令》
ＯＰＥＮ命令は、データファイルを使うとき最初に必要。ＯＰＥＮ命令で、ファイル名とファイル番号を結びつける。

①シーケンシャルファイルを作るとき

```
ＯＰＥＮ”デバイス名：ファイル名”　ＦＯＲ　ＯＵＴＰＵＴ　ＡＳ〔＃〕<ファイル番号>
```

②シーケンシャルファイルを読むとき

```
ＯＰＥＮ”デバイス名：ファイル名”　ＦＯＲ　ＩＮＰＵＴ　ＡＳ〔＃〕<ファイル番号>
```

③シーケンシャルファイルにデータを追加するとき

```
ＯＰＥＮ”デバイス名：ファイル名”　ＦＯＲ　ＡＰＰＥＮＤ　ＡＳ〔＃〕<ファイル番号>
```

④ランダムファイルを読み書きするとき

```
ＯＰＥＮ”デバイス名：ファイル名”　ＡＳ〔＃〕<ファイル番号>
```

《ＣＬＯＳＥ命令》
ＣＬＯＳＥ命令は、データファイルを使い終わったときに必要。ＣＬＯＳＥ命令で、ファイル名とファイル番号は対応しなくなる。

```
ＣＬＯＳＥ　〔〔＃〕<ファイル番号>〔，〔＃〕<ファイル番号>〕・・・〕
```

ファイル番号を省略すると、ＯＰＥＮしていたすべてのファイルをＣＬＯＳＥする。
ＥＮＤ文でも、すべてのファイルをＣＬＯＳＥする。

☆シーケンシャルファイルの入出力☆

《出力命令》

出力命令には、ＰＲＩＮＴ＃とＷＲＩＴＥ＃がある。

①ＰＲＩＮＴ＃

文字項目は、必ずコンマ（，）で区切る。数字項目は、セミコロン（；）で区切ってもよい。
文字項目と数字項目の間は、必ずコンマ（，）で区切る。
文字項目の中にコンマ（，）を入れる時は、ＣＨＲ＄（３４）（”）で囲む。

（例１）　正しい書き方

```
１０　ＯＰＥＮ　”ＴＥＳＴ”　ＦＯＲ　ＯＵＴＰＵＴ　ＡＳ　＃１
２０　Ａ＄＝”ＡＢＣ”：Ｂ＄＝”ＤＥＦ”：Ｃ＝１５０：Ｄ＝－３０
３０　ＰＲＩＮＴ＃１，Ａ＄；”，”；Ｂ＄；”，”；Ｃ；Ｄ
４０　ＣＬＯＳＥ＃１
５０　ＥＮＤ
```

ディスクには、次のように記録される。数字はすべて１６進数。「キャラクター・コード表」参照

41	42	43	2C	44	45	46	2C	20	31	35	30	20	2D	33	30	20	0D	0A	1A
A	B	C	,	D	E	F	,	△	1	5	0	△	-	3	0	△	CR	LF	SB
A＄			区切り	B＄			区切り	C				区切り	D						

レコードの区切りには、ＣＲ（Carriage return）・ＬＦ（Line feed）がつく。
ファイルの終わりには、ＳＢ（Substitute character）がつく。△は、空白を表す。

（例２）各項目の間を、コンマ（，）で区切らないと、１つの項目になってしまう。
（例１）の行番号３０を次のようにする。

```
３０　ＰＲＩＮＴ＃１，Ａ＄；Ｂ＄；Ｃ；Ｄ
```

41	42	43	44	45	46	20	31	35	30	20	2D	33	30	20	0D	0A	1A
A	B	C	D	E	F	△	1	5	0	△	-	3	0	△			

（A〜△まで）項目がつながってしまう

（例３）各項目をコンマ（，）だけでつなぐと、余分な空白が入り１つの項目になってしまう。
（例１）の行番号３０を次のようにする。

```
３０　ＰＲＩＮＴ＃１，Ａ＄，Ｂ＄，Ｃ，Ｄ
```

41	42	43	20‥	44	45	46	20‥	20	31	35	30	20‥	2D	33	30	20	0D	0A	1A
			空白11個				空白11個					空白10個							
全体で１４文字				全体で１４文字				全体で１４文字											

（例４）文字項目の中にコンマ（，）やダブルコーテーション（二重引用符”）を入れるには、ＣＨＲ＄（３４）を使う。
（例１）のプログラムの行番号２０・３０を次のようにする。

```
２０　Ａ＄＝ＣＨＲ＄（３４）＋”ＡＢ，Ｃ”＋ＣＨＲ＄（３４）：Ｂ＄＝”ＤＥ”＋ＣＨ
　　　Ｒ＄（３４）＋”Ｆ”
３０　ＰＲＩＮＴ＃１，Ａ＄；”，”；Ｂ＄
```

22	41	42	2C	43	22	2C	44	45	22	46	0D	0A	1A
”	A	B	,	C	”	,	D	E	”	F			
A＄						区切り	B＄						

②ＷＲＩＴＥ＃
項目の区切りはコンマ（，）でも、セミコロン（；）でもかまわない。
文字項目の中に、ダブルコーテーション（”）を入れられない。

（例１）　正しい書き方

```
１０　ＯＰＥＮ　”ＴＥＳＴ”　ＦＯＲ　ＯＵＴＰＵＴ　ＡＳ　＃１
２０　Ａ＄＝”ＡＢＣ”：Ｂ＄＝”ＤＥＦ”：Ｃ＝１５０：Ｄ＝－３０
３０　ＷＲＩＴＥ＃１，Ａ＄；Ｂ＄；Ｃ；Ｄ　　←（セミコロンのかわりに、コンマでも同じ）
４０　ＣＬＯＳＥ＃１
５０　ＥＮＤ
```

22	41	42	43	22	2C	22	44	45	46	22	2C	31	35	30	2C	2D	33	30	0D	0A	1A
”	A	B	C	”	,	”	D	E	F	”	,	1	5	0	,	−	3	0			
A＄					区切り	B＄					区切り	C			区切り	D					

（例２）文字項目の区切りはダブルコーテーション（”）なので、文字項目の中にダブルコーテーションを入れると、項目を正しく区切らない。
（例１）のプログラムの行番号２０・３０を次のようにする。

```
２０　Ａ＄＝”ＡＢ”＋ＣＨＲ＄（３４）＋”Ｃ”：Ｂ＄＝”ＤＥＦ”
３０　ＷＲＩＴＥ＃１，Ａ＄；Ｂ＄
```

A＄							B＄							
22	41	42	22	43	22	2C	22	44	45	46	22	0D	0A	1A
”	A	B	”	C	”	,	”	D	E	F	”			
項目			区切り	項目		区切り	項目							

読み込むときは３つの項目とみなす。

《入力命令》
シーケンシャルファイルを読むには、ＩＮＰＵＴ＃かＬＩＮＥ　ＩＮＰＵＴ＃のいずれかを使う。

①ＩＮＰＵＴ＃
１レコードを読み込み、項目の区切りごとに対応する変数へ入れる。
（例１）ＩＮＰＵＴ＃で読み込む。

```
１０　ＯＰＥＮ　”ＴＥＳＴ”　ＦＯＲ　ＯＵＴＰＵＴ　ＡＳ　＃１
２０　Ａ＄＝”ＡＢＣ”：Ｂ＄＝”ＤＥＦ”：Ｃ＝１５０：Ｄ＝－３０
３０　ＷＲＩＴＥ＃１，Ａ＄，Ｂ＄，Ｃ，Ｄ
４０　ＣＬＯＳＥ＃１
５０　ＥＮＤ
```

として作ったファイルを読む。

```
１０　ＯＰＥＮ　”ＴＥＳＴ”　ＦＯＲ　ＩＮＰＵＴ　ＡＳ　＃１
２０　ＩＮＰＵＴ＃１，Ａ１＄，Ｂ１＄，Ｃ１，Ｄ１
３０　ＣＬＯＳＥ＃１
４０　ＥＮＤ
```

Ａ＄						Ｂ＄						Ｃ				Ｄ					
22	41	42	43	22	2C	22	44	45	46	22	2C	31	35	30	2C	2D	33	30	0D	0A	1A
	Ａ１＄						Ｂ１＄					Ｃ１				Ｄ１					

②ＬＩＮＥ　ＩＮＰＵＴ＃
１レコードを読み込み、すべて１つの変数へ入れる。
（例２）例１で作ったＴＥＳＴファイルをＬＩＮＥ　ＩＮＰＵＴ＃で読み込む。

```
１０　ＯＰＥＮ　”ＴＥＳＴ”　ＦＯＲ　ＩＮＰＵＴ　ＡＳ　＃１
２０　ＬＩＮＥ　ＩＮＰＵＴ＃１，Ｚ＄
３０　ＣＬＯＳＥ＃１
４０　ＥＮＤ
```

Ａ＄						Ｂ＄						Ｃ				Ｄ					
22	41	42	43	22	2C	22	44	45	46	22	2C	31	35	30	2C	2D	33	30	0D	0A	1A
Ｚ＄																					

アスキーセーブしたプログラムファイルなどを読むときは、ＬＩＮＥ　ＩＮＰＵＴ＃を使えば便利。

《シーケンシャルファイルで使う関数》
①ファイルの終わりを見つける
ファイルの終わりを見つけるには、ＥＯＦ関数を使う。

ＥＯＦ（<ファイル番号>）

（例）ファイルの終わりになれば、＊ＯＷＡＲＩへとぶ。

```
１０　ＯＰＥＮ　”ＵＲＩＡＧＥ”　ＦＯＲ　ＩＮＰＵＴ　ＡＳ　＃１
２０　ＩＦ　ＥＯＦ（１）　ＴＨＥＮ　ＧＯＴＯ　＊ＯＷＡＲＩ
３０　ＩＮＰＵＴ　＃１，Ａ＄，Ｂ
４０　ＰＲＩＮＴ　Ａ＄，Ｂ
５０　ＧＯＴＯ　２０
６０　＊ＯＷＡＲＩ
７０　ＣＬＯＳＥ　＃１
８０　ＥＮＤ
```

②読み書きしたレコード数を求める

読み書きしたレコード数は、ＬＯＣ関数で求める。ただし、１レコードは２５６バイトとみなす。ＬＯＣ関数を使う前に、ファイルをＯＰＥＮしておかねばならない。

```
ＬＯＣ（＜ファイル番号＞）
```

(例) ３レコード書き出したが、ＬＯＣ関数の結果は１になる。

```
１０　ＯＰＥＮ　”ＴＥＳＴ”　ＦＯＲ　ＯＵＴＰＵＴ　ＡＳ　＃１
２０　Ａ＄＝ＳＴＲＩＮＧ＄（４０，”Ａ”）
３０　ＦＯＲ　Ｊ＝１　ＴＯ　３
４０　　　　ＰＲＩＮＴ　＃１，Ａ＄
５０　ＮＥＸＴ　Ｊ
６０　ＰＲＩＮＴ　ＬＯＣ（１）
７０　ＣＬＯＳＥ　＃１
８０　ＥＮＤ
```

41	41……41	0D	0A	41	41……41	0D	0A	41	41……41	0D	0A	1A
Aが４０個				Aが４０個				Aが４０個				

全体で１２７バイト

このファイルで使うのは１２７バイト。２５６バイトを１レコードとみなすので、ＬＯＣの結果は１になる。

③ファイルの大きさを求める

ファイルが使っているセクター数を求めるには、ＬＯＦ関数を使う。１セクターは２５６バイト。シーケンシャルファイルの場合、ＬＯＦとＬＯＣ関数の値は、同じになる。ＬＯＦ関数を使う前には、ファイルをＯＰＥＮしておかねばならない。

```
ＬＯＦ（＜ファイル番号＞）
```

(例) ＬＯＦ・ＬＯＣ関数の結果は、３になる。

```
１０　ＯＰＥＮ　”ＴＥＳＴ”　ＦＯＲ　ＯＵＴＰＵＴ　ＡＳ　＃１
２０　Ａ＄＝ＳＴＲＩＮＧ＄（２５３，”Ａ”）
３０　ＦＯＲ　Ｊ＝１　ＴＯ　３
４０　　　　ＰＲＩＮＴ　＃１，Ａ＄
５０　ＮＥＸＴ　Ｊ
６０　ＰＲＩＮＴ　”セクタースウ＝”；ＬＯＦ（１）
７０　ＰＲＩＮＴ　”レコードスウ＝”；ＬＯＣ（１）
８０　ＣＬＯＳＥ　＃１
９０　ＥＮＤ
```

☆ランダムファイルの入出力☆

《ＦＩＥＬＤ文》

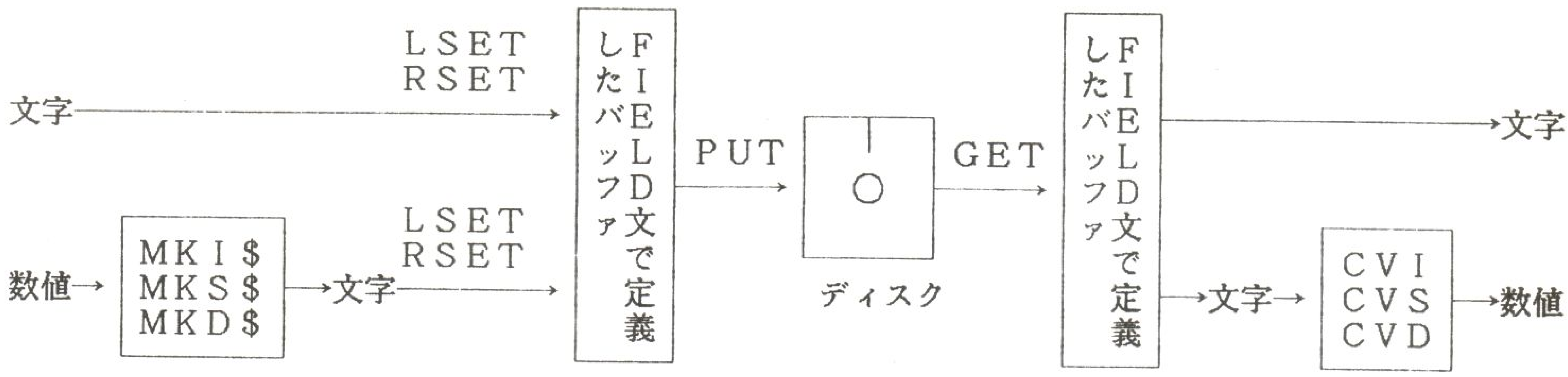
ランダムファイルでは、バッファをとおして入出力しなければならない。２５６バイトのバッファを各項ごとに区切るために、ＦＩＥＬＤ文を使う。

```
ＦＩＥＬＤ〔＃〕＜ファイル番号＞，＜フィールドの大きさ＞　ＡＳ　＜文字変数＞
　　　　　　　　　〔，＜フィールドの大きさ＞　ＡＳ　＜文字変数＞・・・・〕
```

ＦＩＥＬＤ文では、合計２５６バイトまで割り当てられる。

```
ランダムファイルのバッファ＝２５６バイト＝ランダムファイルの１レコードの長さ
```

バッファの中では、すべて文字として扱う。したがって書き出すとき、数値はＭＫＩ＄・ＭＫＳ＄・ＭＫＤ＄のいずれかの関数を使って文字に変換する。読み込んだときは、ＣＶＩ・ＣＶＳ・ＣＶＤ関数を使って文字から数値に変換する。文字をバッファに入れるには、ＬＳＥＴ・ＲＳＥＴ命令を使う。



《出力命令》

ランダムファイルに書き出すには、PUT命令を使う。

```
PUT〔#〕<ファイル番号>,〔<式>〕
```

<式>には、レコード番号を指定する。式を省略すると、最後にPUTかGETした次のレコード番号として書き出す。

(例1) "URIAGE"ファイルを作る。

```
10 OPEN "URIAGE" AS #1
20 FIELD#1,20 AS FLNAMAE$,4 AS FLURIAGE$
30 INPUT "バンゴウ";A%
40 IF A%=0 THEN GOTO *OWARI
50 INPUT "ナマエ";NAMAE$
60 INPUT "ウリアゲ";URIAGE
70 LSET FLNAMAE$=NAMAE$
80 RSET FLURIAGE$=MKS$(URIAGE)
90 PUT #1,A%
100 GOTO 30
110 *OWARI
120 CLOSE #1
130 END
```

FIELD文で定義しなかった残りのバッファには、0が入る。
書き出せるレコード番号の最大値は、フロッピーディスクの残りのセクター数に等しい。

(98) レコード番号の最大値=DSKF(ドライブ番号)×(1クラスターあたりのセクター数)－1
(88) レコード番号の最大値=DSKF(ドライブ番号)×(1クラスターあたりのセクター数)
1クラスターあたりのセクター数=8インチフロッピーなら26。5インチフロッピーなら8

ファイルをOPENすると、自動的に1クラスター分とられる。したがって、OPEN後にDSKFをみると、残りのクラスター数は実際の値より1少なくなるので注意が必要。

(例2) "URIAGE"ファイルを作る。番号は、レコード番号の最大値以下とする。

```
10 L=DSKF(1)*26 ……→8インチの場合
20 OPEN "URIAGE" AS #1
30 FIELD#1,20 AS FLNAMAE$,4 AS FLURIAGE$
40 INPUT "バンゴウ";A%
50 IF A%=0 THEN GOTO *OWARI
60 IF A%>L THEN PRINT "バンゴウ オーバー":GOTO 40
70 INPUT "ナマエ";NAMAE$
80 INPUT "ウリアゲ";URIAGE
90 LSET FLNAMAE$=NAMAE$
100 RSET FLURIAGE$=MKS$(URIAGE)
110 PUT #1,A%
120 GOTO 40
130 *OWARI
140 CLOSE #1
150 END
```

《入力命令》
ランダムファイルを読み込むにはGET命令を使う。

```
GET〔#〕<ファイル番号>,〔<式>〕
```

<式>にはレコード番号を指定する。省略すると、最後にGETした次のレコード番号として読み込む。

（例3）例2で作った"URIAGE"ファイルを読む。

```
10 OPEN "URIAGE" AS #1
20 FIELD#1,20 AS FLNAMAE$,4 AS FLURIAGE$
30 L=LOF(1)
40 INPUT "バンゴウ";A%
50 IF A%=0 THEN GOTO *OWARI
60 IF A%>L THEN PRINT "バンゴウ ナシ":GOTO 40
70 GET #1,A%
80 PRINT "バンゴウ=";A%;"ナマエ=";FLNAMAE$;
90 PRINT "ウリアゲ=";CVS(FLURIAGE$)
100 GOTO 40
110 *OWARI
120 CLOSE #1
130 END
```

《ランダムファイルで使う関数》

①バッファに入れるために、数値を文字列に変換する

```
x=整数 ─────→MKI$(x) →2バイトの文字列に変換
x=単精度実数 →MKS$(x) →4バイトの文字列に変換
x=倍精度実数 →MKD$(x) →8バイトの文字列に変換
```

②バッファに入っている文字列を、数値に変換する。Sは文字式

```
S=2バイトの文字列 →CVI(S) →整数に変換
S=4バイトの文字列 →CVS(S) →単精度実数に変換
S=8バイトの文字列 →CVD(S) →倍精度実数に変換
```

③読み書きしたレコード番号を求める

```
LOC(<ファイル番号>)
```

④ファイルの最大レコード番号を求める

```
LOF(<ファイル番号>)
```

（例4）"URIAGE"ファイルをすべて読む

```
10 OPEN "URIAGE" AS #1
20 FIELD #1,20 AS FLNAMAE$,4 AS FLURIAGE$
30 FOR J=1 TO LOF(1)
40    GET #1
50    PRINT FLNAMAE$;CVS(FLURIAGE$)
60 NEXT J
70 CLOSE #1
80 END
```

《FIELD文で配列定義》

ランダムファイルでは、1レコードが256バイトで一定なので、書き出す項目の長さが短いと、使わない無駄な場所が増える。また、配列をそのまま書き出せない。このようなときには、FIELD文を使ってバッファを配列のように定義する。

（例）名前と売り上げを配列定義して書き出す。バッファは以下のように定義する。

名前（0）	売り上げ（0）	名前（1）	売り上げ（1）……名前（9）	売り上げ（9）

```
10 FOR J=0 TO 9
20    FIELD#1,J*24 AS DUMMY$,20 AS FLNAMAE$(J)
              ,4 AS FLURIAGE$(J)
30 NEXT J
40 OPEN "URIAGE" AS #1
50 C=0
60 INPUT "ナマエ";NAMAE$
70 IF NAMAE$="" THEN GOTO *OWARI
80 INPUT "ウリアゲ";URIAGE
90 LSET FLNAMAE$(C)=NAMAE$
100 RSET FLURIAGE$(C)=MKS$(URIAGE)
110 C=C+1
120 IF C<>10 THEN GOTO 60
130 PUT #1
140 GOTO 50
150 *OWARI
160 IF C=0 THEN GOTO 220
170 FOR J=C TO 9
180    LSET FLNAMAE$(J)=""
190    RSET FLURIAGE$(J)=MKS$(0)
200 NEXT J
210 PUT #1
220 CLOSE #1:END
```

☆その他のＩ／Ｏ命令☆

《ＩＮＰＵＴ＄》

ファイルから、指定した数の文字列を読む。シーケンシャルファイルでもランダムファイルでもよい。

ＩＮＰＵＴ＄（＜文字数＞〔，〔＃〕＜ファイル番号＞〕） ただし、文字数は２５５以下。

《ＤＳＫＩ＄とＤＳＫＯ＄》

ＤＳＫＩ＄は、指定したセクターを０番のバッファと指定した変数に読み込む。ＤＳＫＯ＄は、０番のバッファの内容を指定したセクターへ書き出す。どちらの命令も、ＯＰＥＮ・ＣＬＯＳＥは必要ない。

（1）ＤＳＫＩ＄（関数）

8インチ・5インチ両面フロッピーのとき
ＤＳＫＩ＄（＜ドライブ番号＞，＜サーフェス番号＞，＜トラック番号＞，＜セクター番号＞）
5インチ片面フロッピーのとき
ＤＳＫＩ＄（＜ドライブ番号＞，＜トラック番号＞，＜セクター番号＞）

ＤＳＫＩ＄の結果を文字変数に入れるときは、２５５バイトまでしか入らないので注意。

（2）ＤＳＫＯ＄

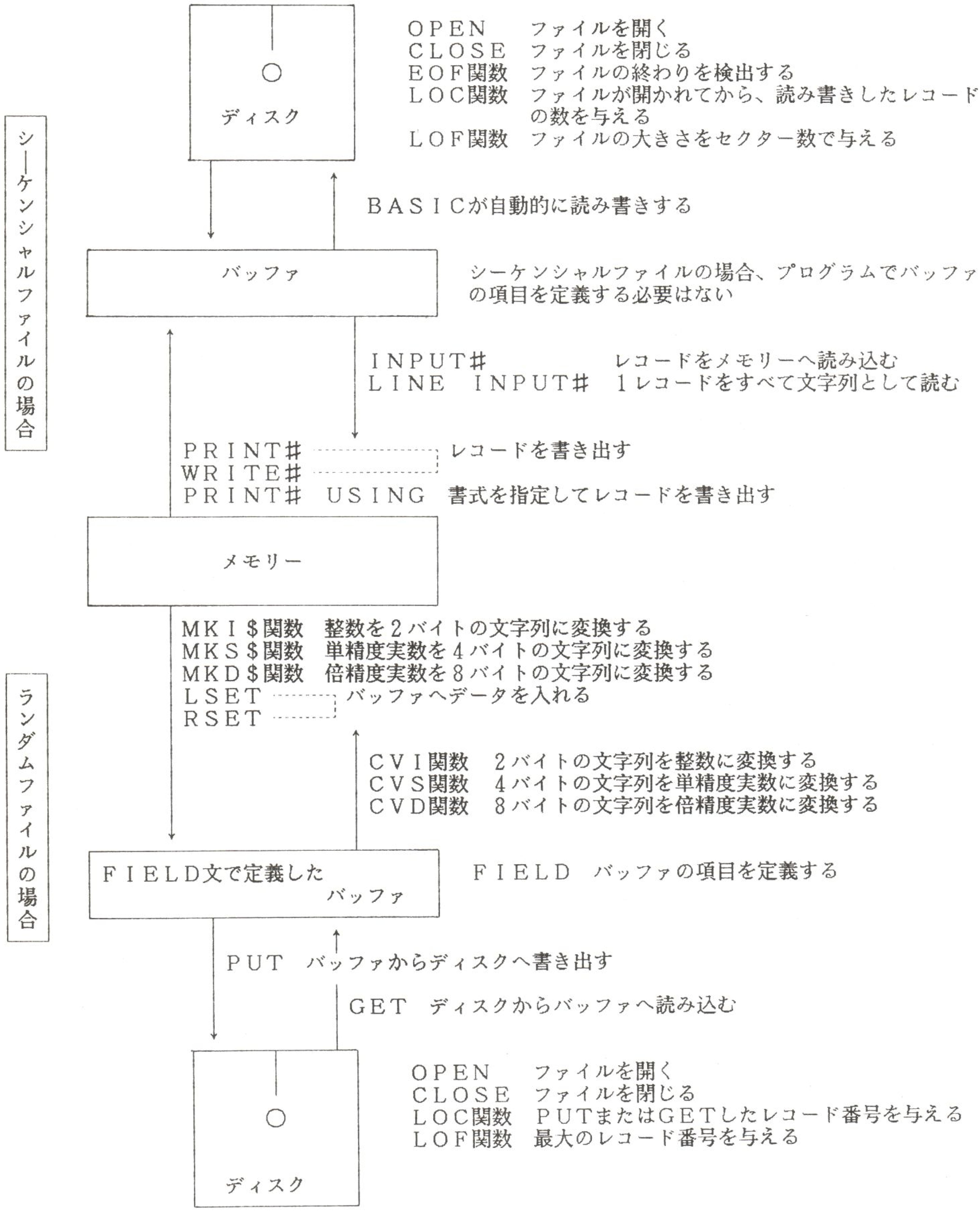
8インチ・5インチ両面フロッピーのとき
ＤＳＫＯ＄　＜ドライブ番号＞，＜サーフェス番号＞，＜トラック番号＞，＜セクター番号＞
5インチ片面フロッピーのとき
ＤＳＫＯ＄　＜ドライブ番号＞，＜トラック番号＞，＜セクター番号＞

ＤＳＫＯ＄で書き出すときは、バッファの０番をＦＩＬＥＤ文で定義して使う。

（例）ドライブ3番に入っている5インチ両面フロッピーをドライブ4番にコピーする。

```
10 FIELD #0,128 AS A1$,128 AS A2$
20 FOR M=0 TO 1
30    FOR T=0 TO 39
40       FOR S=1 TO 2*8
50          Z$=DSKI$(3,M,T,S)
60          DSKO$ 4,M,T,S
70       NEXT S
80    NEXT T
90 NEXT M
100 END
```

☆フロッピーディスクの命令一覧☆



その他の入出力命令

INPUT$（文字数，ファイル番号）	ファイルから指定した文字数を読む
DSKI$・DSKO$	指定したセクターを読み書きする

☆ディスクファイルの管理☆

フロッピーディスクの読み書きは、１セクター単位で行われる。したがって、サーフェス番号（両面のとき）、トラック番号とセクター番号で物理的に読み書きする場所が決められる。通常は、特別に意識しなくても、ファイル名さえ与えれば、自動的に読み書きする。管理は、１クラスター単位でＢＡＳＩＣが行っている。標準フロッピーディスクの場合、１クラスターは２６セクター、ミニフロッピーディスクの場合、１クラスターは8セクターである。

《クラスターとサーフェス、トラック、セクターの関係》

	８インチ両面倍密度			５インチ両面倍密度		
クラスター	サーフェス	トラック	セクター	サーフェス	トラック	セクター
０	０	０	１～２６	０	０	１～　８
１	１	０	１～２６	０	０	９～１６
２	０	１	１～２６	１	０	１～　８
３	１	１	１～２６	１	０	９～１６
４	０	２	１～２６	０	１	１～　８
⋮	⋮	⋮	⋮	⋮	⋮	⋮
１５３	１	７６	１～２６	⋮	⋮	⋮
⋮						
１５９				１	３９	９～１６

サーフェス
０…おもて面
１…うら面

５インチ片面倍密度のとき
クラスターは
０～６９
トラックは
０～３４

《トラックの使用割り当て》
（１）８インチ両面倍密度
＊システムディスク（ＤＩＳＫ－ＢＡＳＩＣが記憶されているディスク）
（ＰＣ－９８０１）

サーフェス	トラック	セクター	内　　容
０	０	１～４	ＩＰＬ
	０	５～２６	使えない
	１～２	すべて	ＤＩＳＫ－ＢＡＳＩＣ
	３～３４	すべて	ユーザーが使える領域
	３５	１～２２	ディレクトリー
	３５	２３	ＩＤ
	３５	２４～２６	ＦＡＴ
	３６～７６	すべて	ユーザーが使える領域
１	０	すべて	使えない
	１～２	すべて	ＤＩＳＫ－ＢＡＳＩＣ
	３～７６	すべて	ユーザーが使える領域

（ＰＣ－８８０１のとき）

・トラック０はすべて使えない
・トラック1のセクター１はＩＰＬ
・トラック１のセクター２～２６とトラック２はＤＩＳＫ－ＢＡＳＩＣが入っている
（上記以外は左表と同じ）

・トラック０は同じ
・トラック１はＤＩＳＫ－ＢＡＳＩＣが入っている
・トラック２～７６はすべてユーザーが使える領域

＊データディスク（ＤＩＳＫ－ＢＡＳＩＣを含まないディスク）

サーフェス	トラック	セクター	内　　容
０	０	すべて	使えない
	１～３４	すべて	ユーザーが使える領域
	３５	１～２２	ディレクトリー
	３５	２３	ＩＤ
	３５	２４～２６	ＦＡＴ
	３６～７６	すべて	ユーザーが使える領域
１	０	すべて	使えない
	１～７６	すべて	ユーザーが使える領域

(２) ５インチ両面倍密度

＊システムディスク（ＤＩＳＫ－ＢＡＳＩＣが記憶されているディスク）
（ＰＣ－９８０１）

サーフェス	トラック	セクター	内　　容
０	０	１～２	ＩＰＬ
	０	３～１６	ＤＩＳＫ－ＢＡＳＩＣ
	１～３	すべて	ＤＩＳＫ－ＢＡＳＩＣ
	４～３９	すべて	ユーザーが使える領域
１	０～３	すべて	ＤＩＳＫ－ＢＡＳＩＣ
	４～１７	すべて	ユーザーが使える領域
	１８	１～１２	ディレクトリー
	１８	１３	ＩＤ
	１８	１４～１６	ＦＡＴ
	１９～３９	すべて	ユーザーが使える領域

（ＰＣ－８８０１のとき）

・トラック０のセクター１はＩＰＬ
・トラック０のセクター２～１６とトラック１はＤＩＳＫ－ＢＡＳＩＣが入っている
・トラック２～３９はユーザーが使える領域
・トラック０～１はＤＩＳＫ－ＢＡＳＩＣが入っている
・トラック２～１７はユーザーが使える領域

（上記以外は左表と同じ）

＊データディスク（ＤＩＳＫ－ＢＡＳＩＣを含まないディスク）

サーフェス	トラック	セクター	内　　容
０	０～３９	すべて	ユーザーが使える領域
１	０～１７	すべて	ユーザーが使える領域
	１８	１～１２	ディレクトリー
	１８	１３	ＩＤ
	１８	１４～１６	ＦＡＴ
	１９～３９	すべて	ユーザーが使える領域

(３) ５インチ片面倍密度

＊システムディスク（ＤＩＳＫ－ＢＡＳＩＣが記憶されているディスク）
（ＰＣ－９８０１）

トラック	セクター	内　　容
０	１～２	ＩＰＬ
０	３～１６	ＤＩＳＫ－ＢＡＳＩＣ
１～６	すべて	ＤＩＳＫ－ＢＡＳＩＣ
７～１７	すべて	ユーザーが使える領域
１８	１～１２	ディレクトリー
１８	１３	ＩＤ
１８	１４～１６	ＦＡＴ
１９～３４	すべて	ユーザーが使える領域

（ＰＣ－８８０１のとき）

・トラック０のセクター１はＩＰＬ
・トラック０のセクター２～１６とトラック１～３はすべてＤＩＳＫ－ＢＡＳＩＣが入っている
・トラック４～１７はユーザーが使える領域

（上記以外は左表と同じ）

＊データディスク（ＤＩＳＫ－ＢＡＳＩＣを含まないディスク）

トラック	セクター	内　　容
０～１７	すべて	ユーザーが使える領域
１８	１～１２	ディレクトリー
１８	１３	ＩＤ
１８	１４～１６	ＦＡＴ
１９～３４	すべて	ユーザーが使える領域

《ディレクトリー》
　ディレクトリーは、ディスクに記録されているファイルの名前、属性、どの場所から記憶しているかを示す登録簿。ファイルごとに１つ作られ、各ディレクトリーは１６バイトとられる。
〔ディレクトリーの構造〕

バイト	内　　　容
０～５	ファイル名　先頭バイトが１６進数のＦＦならば、未使用のディレクトリー ００ならば、削除されたファイル
６～８	拡張子
９	属性を示す ００：アスキー形式 ０１：機械語形式 １０：アスキー形式、書き込み禁止 ４０：アスキー形式 　　　リード・アフター・ライト ８０：非アスキー形式 ９０：非アスキー形式、書き込み禁止 Ａ０：暗号化された非アスキー形式 Ｃ０：非アスキー形式 　　　リード・アフター・ライト
１０	ファイルの先頭クラスター番号
１１～１５	未使用

《ＦＡＴ》
　ＦＡＴ（ファイル・アロケーション・テーブル）は、各クラスターの使用・未使用状態を示す。ファイルは、物理的につながったクラスター上に作られるわけではなく、離れたクラスターをつないで構成される。ディスクを最初にアクセスしたときにＦＡＴがメモリーにコピーされる。ＦＡＴは、３セクターにわたってとられ、それぞれのセクターの内容はすべて同一となっている。１バイトでクラスターの使用状態を表す。

ＦＡＴの値	クラスターの状態
8インチ両面：0～99 5インチ両面：0～9F 5インチ片面：0～45	使用中を示す。後続クラスターの番号が入っている
8インチ　：C0～DA 　（88：C1～DA） 5インチ　：C0～C8 　（88：C1～C8）	使用中を示す。連続するクラスターの最後。8インチのとき下位5ビット、5インチのとき下位4ビットが、そのクラスターで実際に使われているセクター数が入っている
ＦＥ	使えない。ＤＩＳＫ－ＢＡＳＩＣやディレクトリーなどで使っている
ＦＦ	未使用

《ＩＤ》
　ＩＤは、１セクター（２５６バイト）でディスクの属性などを示す。

バイト	内　　　容
０	属性を示す。ディレクトリーの属性と同じ
１	ファイル数を指定する（システムディスクにのみ有効） ＦＦ　　：システムディスクでＤＩＳＫ－ＢＡＳＩＣスタート時に 　　　　　Ｈｏｗ　ｍａｎｙ　ｆｉｌｅｓ（０－１５）？の表示が出る ０～１５：上記のメッセージが出ず、この数が同時にオープンするファイルの数として設定される
２～２５５	ＢＡＳＩＣテキスト（システムディスクにのみ有効） 　ここに設定した内容がＤＩＳＫ－ＢＡＳＩＣスタート後実行される

　ファイル数、ＢＡＳＩＣテキストの設定の仕方は、「プログラムの自動スタート」参照。

☆ユーティリティープログラムの使い方☆

《プログラムの自動スタート》
　ＤＩＳＫ－ＢＡＳＩＣのスタート後、自動的にＢＡＳＩＣの命令を実行させるために、ＩＤセクターのファイル数、ＢＡＳＩＣのテキストを書き換える。ＩＤセクターを書き換えるには、システムディスクにはいっているｓｅｔｉｎｆ．ｎ８８プログラムを使う。ＢＡＳＩＣのテキストを”ＲＵＮ<プログラム名>”とすると、指定したプログラムを自動的に実行する。
〔ｓｅｔｉｎｆ．ｎ８８プログラムの実行〕

（1）システムディスクからｓｅｔｉｎｆ．ｎ８８プログラムをロードし、実行する。

（2）

```
Set up info sector
Mount a system disk on drive#?‥‥→（98）
```

ＩＤセクターの書き換えをしようとするシステムディスクを入れ、ドライブ番号【ＲＥＴ】。

```
Sure (y/n) ?  (98)
```

確認後、ドライブ番号が正しければ、ｙ【ＲＥＴ】。
まちがっていれば、ｎ【ＲＥＴ】で（2）へもどる。

（3）

```
How many files (0-15) ?
```

同時にオープンするファイルの数（０～１５）を指定する。－１を入力すると、ＢＡＳＩＣスタート時にＨｏｗ　ｍａｎｙ　ｆｉｌｅｓ（０－１５）？の問い合わせがあり、ＢＡＳＩＣのテキストは無効になる。（８８）は、有効。

（4）

```
Text for init?
```

ＤＩＳＫ－ＢＡＳＩＣスタート後、実行したいコマンドやステートメントを入力する。自動的に指定したプログラムを走らせたいときには、ＲＵＮ”プログラム名”【ＲＥＴ】と指定する。コマンドやステートメントを消したいときは、【ＲＥＴ】。コマンドやステートメントが２５３文字を越えると下記のメッセージが表示され、（4）へもどる。

```
Text can have a maximum of 253 characters
```

（5）

```
#of files to allocate is ①
 Text for init is ②
 Mount a system disk on drive# ③
 Type RETURN if ready
```

①は（3）で入力した数
②は（4）で入力したコマンドやステートメント
③は（2）で入力したドライブ番号。（８８）のときは、ドライブ番号1
（3）で－１を入力したときには、以下の表示がでる。

```
#of files is asked when starting up
```

ＩＤセクターの書き換えをしようとするシステムディスクをドライブ番号③のドライブに装着し【ＲＥＴ】。

（6）

```
All done.
```

を表示し、終了。

《ファイルの複写》
　フロッピーディスク間でファイルの複写をするには、システムディスクに入っているｘｆｉｌｅｓ．ｎ８８プログラムを使う。フロッピーディスクの種類が異なってもよい。ただし、ＤＩＳＫ－ＢＡＳＩＣは複写しない。複写先に同じ名前のファイルがあると、複写元のファイルに置き換わる。

〔ｘｆｉｌｅｓ．ｎ８８プログラムの実行〕

（1）システムディスクからｘｆｉｌｅｓ．ｎ８８プログラムをロードし、実行する。

（2）

```
Ｔｒａｎｓｆｅｒ　ｆｉｌｅｓ
Ｆｒｏｍ　ｄｒｉｖｅ＃？
```

複写元のフロッピーをドライブに装着し、ドライブ番号【ＲＥＴ】。

（3）

```
Ｔｏ　ｄｒｉｖｅ＃？
```

複写先のフロッピーをドライブに装着し、ドライブ番号【ＲＥＴ】。

（4）

```
Ｓｕｒｅ（ｙ／ｎ）？
```

確認して正しければ、ｙ【ＲＥＴ】。まちがっていれば、ｎ【ＲＥＴ】で(2)へもどる。
（８８）はｎ【ＲＥＴ】（４）へもどる。

```
Ｃｏｐｙｉｎｇ　ｆｉｌｅ　①　②→③
```

を表示し、ファイルを転送する。
①はファイル名
②は転送する側のドライブ番号
③は転送される側のドライブ番号

（5）

```
Ｃｏｍｐｌｅｔｅｄ．
```

を表示し終了。

《ＤＩＳＫ－ＢＡＳＩＣだけの複写》

ＤＩＳＫ－ＢＡＳＩＣがこわれてしまったとか、バージョンアップしたとかの理由で、ＤＩＳＫ－ＢＡＳＩＣだけ複写したいとき、システムディスクに入っているｓｙｓｇｅｎ．ｎ８８プログラムを使う。（８８）のときはｓｙｓｇｅｎプログラムを使う。

〔ｓｙｓｇｅｎ．ｎ８８プログラムの実行〕

（1）システムディスクからｓｙｓｇｅｎ．ｎ８８プログラムをロードし、実行する。

（2）

```
Ｃｏｐｙ　ｓｙｓｔｅｍ
Ｃｏｐｙ　ｆｒｏｍ　ｄｒｉｖｅ＃？
```

複写元のＤＩＳＫ－ＢＡＳＩＣが入っているディスクを入れ、ドライブ番号【ＲＥＴ】

（3）

```
Ｔｏ　ｄｒｉｖｅ＃？
```

複写先のフロッピーを入れ、ドライブ番号【ＲＥＴ】。

（4）

```
Ｈｉｔ　ＲＥＴＵＲＮ　ｉｆ　ｒｅａｄｙ
Ｈｉｔ　ＲＥＴＵＲＮ　ｉｆ　ｙｏｕ　ａｒｅ　Ｏ．Ｋ．　――→（８８）
```

確認して正しければ、【ＲＥＴ】。まちがっていれば、【ＳＴＯＰ】し、再度実行しなおす。

```
Ｗｏｒｋｉｎｇ
```

を表示し、ＤＩＳＫ－ＢＡＳＩＣを複写する。

（5）

```
Ｃｏｍｐｌｅｔｅｄ．
```

を表示し、終了。

14．BASICの数値表現

ＢＡＳＩＣで使う数値には、整数型、単精度実数型、倍精度実数型の３種類ある。整数型は２バイト、単精度は４バイト、倍精度は８バイトで表現する。

☆整数型☆

整数型は、２バイト（＝１６ビット）の２進数で、－３２７６８～３２７６７の整数値を表す。

＜メモリー上の整数表現＞

１バイト目 下位８ビット	２バイト目 上位８ビット
$b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$	$s\ b_{14}\ b_{13}\ b_{12}\ b_{11}\ b_{10}\ b_9\ b_8$

⇩

＜整数の２進数表現＞

上位８ビット		下位８ビット
s	$b_{14}\ b_{13}\ b_{12}\ b_{11}\ b_{10}\ b_9\ b_8$	$b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$

整数型は、図のように符号ビットｓとb14～ｂ０のビットからなる。メモリー上は、上下の８ビットが入れかわる。各ビットは、それぞれ２の累乗に対応する。そして、１になっているビットに対応する累乗値を加え合わせた値が整数値になる。最上位ビットｓには２の１５乗の負数を割りあてる。負の数を表すときｓは１、正の数は０になっている。このような負数の表しかたを２の補数方式と呼ぶ。

ｂ０	$2^{0}=1$
ｂ１	$2^{1}=2$
・	・
ｂ13	$2^{13}=8192$
ｂ14	$2^{14}=16384$
ｓ	$-2^{15}=-32768$

（メモリー上の整数表現を１０進数になおす方法）
メモリー上で

００００１０１０	１００００００００

なら，上位と下位のバイトを入れかえて

１００００００００	００００１０１０

(s … b_3, b_1)

１のビットに対応した２の累乗を加えて

$$-2^{15}+2^{3}+2^{1}=-32758$$

メモリー上の表現値と１６進、２進、１０進数の対応はつぎのとおり。

メモリー上の表現値	１６進数	２　進　数	１０進数
００　００	００００	００００００００００００００００	０
０１　００	０００１	００００００００００００００００１	１
・	・	・	・
ＦＥ　７Ｆ	７ＦＦＥ	０１１１１１１１１１１１１１１０	３２７６６
ＦＦ　７Ｆ	７ＦＦＦ	０１１１１１１１１１１１１１１１	３２７６７
００　８０	８０００	１０００００００００００００００	－３２７６８
０１　８０	８００１	１００００００００００００００１	－３２７６７
・	・	・	・
ＦＥ　ＦＦ	ＦＦＦＥ	１１１１１１１１１１１１１１１０	－２
ＦＦ　ＦＦ	ＦＦＦＦ	１１１１１１１１１１１１１１１１	－１

☆単精度実数型☆

《浮動小数点形式》

実数（1．5や3．14159など）は次のように仮数と指数を使った浮動小数点形式で表す。

$$実数 = \pm (2進数で表した仮数)_2 \times 2^{指数}$$

2進数は $(\cdots\cdots)_2$ で表す。

10進数の0．5は $+(1.0)_2 \times 2^{-1}$ となる。仮数は1．0、指数は－1である。

（1．0 の「1」…仮数の整数部、「0」…仮数の小数部）

《メモリー上の単精度実数型の表現》

仮数24ビットで1ビットが符号、指数は8ビット。各ビットの構成は図のとおり。

1バイト目	2バイト目	3バイト目	4バイト目
$b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$	$b_{15}\ b_{14}\ b_{13}\ b_{12}\ b_{11}\ b_{10}\ b_9\ b_8$	$s\ b_{22}\ b_{21}\ b_{20}\ b_{19}\ b_{18}\ b_{17}\ b_{16}$	$e_7\ e_6\ e_5\ e_4\ e_3\ e_2\ e_1\ e_0$

⇩

指　数	仮　数	
$e_7\ e_6\ e_5\ e_4\ e_3\ e_2\ e_1\ e_0$	s	$b_{22}\ b_{21}\ b_{20}\ b_{19}\ b_{18}\ b_{17}\ b_{16}\ b_{15}\ b_{14}\ b_{13}\ b_{12}\ b_{11}\ b_{10}\ b_9\ b_8\ b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$

（s…符号ビット）

《仮　数》

10進数の0．5の2進数での表し方は、$(0.1)_2 \times 2^0$、$(0.01)_2 \times 2^1$ などいく通りもある。表し方を統一するため、仮数の整数部が1になるように桁合わせする。これを正規化という。メモリー上では仮数の整数部の1を省略して小数部のみを絶対値で表す。

単精度実数型の仮数の絶対値は23ビットからなる。各ビットに次の図のように2の累乗をわりあてる。仮数のビットのうち、1になっているビットに対応する値を加え合わせる。それに1．0を足した値が仮数値になる。

符号ビットは仮数の符号を表す。0なら正、1なら負。

仮数のビット	ビットに割り振った値
b22	2^{-1} ＝0．5＝1／2
b21	2^{-2} ＝0．25＝1／4
b20	2^{-3} ＝0．125＝1／8
b19	2^{-4} ＝0．0625＝1／16
・	・
・	・
b0	2^{-23} ＝0．0000001192・・・

仮数値＝メモリー上の仮数値＋1．0

《指　数》
指数は８ビットの２進数で、－１２８から１２６まで表す。メモリー上の指数は実際の値に１２９（&Ｈ８１）を加えた値である。ゼロは指数をゼロとして表す。

指数値＝メモリー上の指数値－１２９

実際の指数値	＋１２６	＋１	０	－１	－１２８
メモリー上の表現	２５５	１３０	１２９	１２８	１

《単精度実数表現を１０進数にする》
メモリー上の単精度実数表現を１０進数に変換する方法は次のとおり。

（例）

１バイト目	２バイト目	３バイト目	４バイト目	
００００００００	００００００００	１１１１００００	１００００００００	（メモリー上の表現）
下位８ビット	中位８ビット	上位８ビット	指数８ビット	

⇩

１００００００００	１	１１１０００００００００００００００００００００
指数	符号ビット	仮数の絶対値

仮数値の絶対値＝ $(0.1110\cdots + 1.0)_2 = (1.111)_2 = 1 \times 2^{0}+1 \times 2^{-1}+1 \times 2^{-2}+1 \times 2^{-3}$

$= 1 + 0.5+0.25+0.125 = 1.875$

指数値＝ $(10000000)_2 - 129 = 128 - 129 = -1$

符号ビットは１だから仮数値はマイナス

例のメモリー上の値は

$-(1.111)_2 \times 2^{128-129} = -1.875 \times 2^{-1} = -1.875 \times 0.5 = -0.9375$

《１０進数を単精度実数表現にする》

１０進数をメモリー上の単精度実数表現にする方法は次のとおり。

（例）　－０．７５を単精度実数表現にする

$-0.75 = -(0.5 + 0.25) = -(0.11)_2 \times 2^{0} = -(1.1)_2 \times 2^{-1}$

仮数値の絶対値　＝ $(0.1000\cdots + 1.0)_2$

指数値　＝ $-1 + 129 = 128 = (10000000)_2$

負の数値なので符号ビットは１

符号ビット	仮数の絶対値	指数
1	10000000000000000000000	10000000

メモリー上の表現は次のとおり

1バイト目	2バイト目	3バイト目	4バイト目
00000000	00000000	11000000	10000000
下位8ビット	中位8ビット	上位8ビット	指数8ビット

☆倍精度実数型☆

《メモリー上の倍精度実数型の表現》
倍精度実数型では、仮数が56ビットで1ビットが符号。指数が8ビットで実数型を表す。
各ビットの構成は下図のとおり。
メモリー上の数値の意味は、単精度実数型と同じ。

1バイト目						7バイト目	8バイト目
b b 7…0	b b 15…8	b b 23…16	b b 31…24	b b 39…32	b b 47…40	s b b b b b b b 54535251504948	e e e e e e e e 7 6 5 4 3 2 1 0

⇩

指数	符号ビット	仮数
e e e e e e e e 7 6 5 4 3 2 1 0	s	b b b b b b b b ……… b b b b b b b b 5453525150494847 ……… 7 6 5 4 3 2 1 0

15．モニターの使い方

☆モニターの概要☆

割り込み処理ルーチンを作ったり、特殊な周辺装置をつなぐ場合、アセンブラーでプログラムを作った方が効果的である。モニターはアセンブラープログラムを作る手助けをする。この他に、メモリーやディスクの内容を調べたり変更したりできる。ただし、モニターを有効に使うにはＢＡＳＩＣプログラムの内部型や扱うデータの形式、アセンブラーやＩ／Ｏポートの意味などを理解していなければならない。

モニターコマンドは大きく６つの機能に分かれる。

《モニター自体の動き方を変える》

・セグメントベース番地を変える	Ｃコマンド
・ＢＡＳＩＣモードに戻る	＾Ｂコマンド
・扱う数値を１６進数にするか、８進数にするかを決める	Ｂコマンド
・Ｄ・Ｌ・＾Ｄコマンドで画面だけでなくプリンターにも出力する	Ｐコマンド
・コマンドの種類とパラメーターの一覧表を表示する	＾Ａコマンド
・実装しているメモリーが正常かどうかを調べる	ＴＭコマンド

《アセンブラープログラムを扱う》

・アセンブラープログラムを入力する	Ａコマンド
・逆アセンブルして表示する	Ｌコマンド
・機械語プログラムを実行する	Ｇコマンド

ＢＡＳＩＣと比較すればＬコマンドはＬＩＳＴコマンドに、ＧコマンドはＲＵＮコマンドに対応する。

（９８）のＲＯＭ－ＢＡＳＩＣでは、ＡコマンドとＬコマンドは使えない。

《メモリーの内容を操作する》

・メモリーの内容を表示する	Ｄコマンド、Ｅコマンド、Ｓコマンド
・メモリーの内容をキーボードを使って変更する	Ｅコマンド、Ｓコマンド
・メモリーに定数を埋め込む	Ｆコマンド
・メモリー間で内容のコピーをする	Ｍコマンド
・レジスター・フラグの内容を表示・変更する	Ｘコマンド

Ｄ・Ｅ・Ｓコマンドは似た働きをするが、それぞれ特徴を持っている。Ｅ・Ｓコマンドは内容の表示だけでなく、変更もできる。Ｄコマンドは表示しかできないが、プリントモードでプリンターにも出力できる。Ｓコマンドでは１バイトずつ表示・変更する。ＥコマンドではＢＡＳＩＣでプログラムをスクリーンエディットするように、メモリーの内容を表示・変更する。カーソル移動キー・スクロールアップキー・スクロールダウンキーの使い方もＢＡＳＩＣモードと同じである。Ｓコマンドではメモリーだけでなく、メモリースイッチの表示・変更もできる。

《カセットテープを使う》

・カセットテープにメモリーの内容をセーブする	Ｒコマンド
・カセットテープからメモリーにデータをロードする	Ｗコマンド
・カセットテープのデータとメモリーの内容を比較する	Ｖコマンド

カセットテープには番地付きでメモリーの内容をセーブするので、Ｒコマンドを使えば元の番地にデータをロードする。カセットへの転送速度は指定したデバイス名によって決まる。

１：＜ファイル名＞ ……… ６００ボー
２：＜ファイル名＞ ……… １２００ボー

デバイス名を省略した場合はデバイス１が省略値になるので６００ボーである。

《ディスクを扱う》

・ディスク上の内容を表示する	＾Ｄコマンド
・ディスクにメモリーの内容をセーブする	＾Ｒコマンド
・ディスクからメモリーにデータをロードする	＾Ｗコマンド

ディスク上のデータには、番地に関する情報が付いていないので、データをロードする時にはメモリー上のロード開始番地を指定しなければならない。逆に言えば、ロード開始番地を指定するので、どこへでもデータをロードすることができる。

《Ｉ／Ｏポートを扱う》

・入力ポートに値を表示する	Ｉコマンド
・出力ポートに１バイトデータを出力する	Ｏコマンド

☆機械語プログラムセグメント☆

モニターが処理の対象とするセグメントを機械語プログラムセグメントと呼ぶ。Ｃコマンド以外のコマンドで指定する番地は、すべて機械語プログラムセグメント内の相対番地（オフセット）である。モニター使用中は、Ｃコマンドを使って機械語プログラムセグメントのセグメントベース番地を指定する。Ｃコマンドで指定したセグメントベース番地は、次にＣコマンドで再指定するか、モニターを抜けるまで有効である。

機械語モニターに入った直後のセグメントベース番地は次の規則で決まる。

１．ＢＡＳＩＣのＤＥＦ　ＳＥＧ文でセグメントベース番地を指定していれば、それがセグメントベース番地になる。

２．ＣＬＥＡＲ文でＢＡＳＩＣインタープリター領域の上限を指定していれば、それがセグメントベース番地になる。

（注意）

ＣＬＥＡＲ文もＤＥＦ　ＳＥＧ文もＣコマンドも実行せずに他のコマンドを使うと、セグメントベース番地はＲＡＭの上限になる。いずれかの方法で必ずセグメントベース番地を指定すること。

またＢＡＳＩＣインタープリター領域内にセグメントベースを設定した場合、内容を参照するだけならよいが、内容を変更する時は十分確認する必要がある。不用意に内容を変更すると、ＢＡＳＩＣモードに戻った時にプログラムが壊れたり、ＢＡＳＩＣが誤動作する原因になる。

```
ＤＥＦ　ＳＥＧ〔＝＜セグメントベースの値＞〕

ＣＬＥＡＲ〔＜ストリング領域の大きさ＞〕〔，＜ＢＡＳＩＣインタープリター領域の上限＞
　　　　　　　　　　　　〔，＜スタックの大きさ＞〔，＜配列データ領域の大きさ＞〕〕〕
```

☆モニターコマンド一覧表☆

モニターで使えるコマンドは、ＰＣ－９８０１とＰＣ－８８０１では多少異なる。次のページ以降の一覧表はＰＣ－９８０１用なので、ＰＣ－８８０１の場合は、下の相違点を参照した上で使う。

《ＰＣ－８８０１では》

ＰＣ－９８０１はＣＰＵとしてインテルのＩＡＰＸ８６相当のマイクロプロセッサーを使っている。一方ＰＣ－８８０１はμＰＤ７８０を使っている。この２つのマイクロプロセッサーは、使う機械語が異なるばかりでなく、レジスター・フラグの構成、番地の指定方法なども異なっている。

ＰＣ－８８０１のモニターコマンドの体系はＰＣ－９８０１と同じであるが、マイクロプロセッサーの違いに関する部分は、とうぜん違ってくる。

Ａコマンド	使い方は同じであるが、機械語ニーモニックがまったく異なる
Ｃコマンド	ＰＣ－８８０１にはセグメントベースという概念がないので、このコマンドはない
Ｓコマンド	ＰＣ－８８０１にはメモリースイッチがないので、ＳＳＷ〔｛１～７｝〕という形式はない
Ｘコマンド	レジスター・フラグともにＰＣ－９８０１と異なるので、指定の方法が違う。フラグについては個々にではなくＦという名称で一括して扱う ＜ＣＰＵレジスター＞：Ａ，Ｂ，Ｄ，Ｈ，Ａ’，Ｂ’，Ｄ’，Ｈ’，Ｓ，Ｐ，Ｘ，Ｙ，Ｉ ＜ＣＰＵフラグ名＞　：Ｆ レジスターとしてＡ・Ａ’のいずれかを指定した場合は、値を変更する必要のないときでも必ず元の値を入れなければならない。何も入力せずにリターンあるいはスペースキーを押すと、元の内容が壊れる

コマンド	意　　味	入　　力　　形　　式
A	アセンブル	a〔<格納開始番地>〕（入力するときは小文字でよい。以下同じ） アセンブラープログラム入力の状態にする ストップキーまたは^Cを押すか、何も入力せずにリターンキーを押せばアセンブラーから抜ける
B	ベース	b｛hまたはq｝ モニターで扱う数値を16進数にするか、8進数にするか決める h：16進数　q：8進数
C	チェンジ セグメントベース	c〔<セグメントベース番地>〕 機械語セグメントベース番地をかえる
D	ダンプ・メモリー	d〔<表示開始番地>〕〔，<表示終了番地>〕 メモリーの内容を画面に表示する、プリントモードになっていればプリンターにも出力する　（注1）
E	エディット ・メモリー	e〔<エディット開始番地>〕 エディットメモリーモードにする 画面に16×16バイト分のメモリーの内容を表示する。表示内容を書き替えればメモリーの内容も書き替わる ロールアップキー・ロールダウンキーを使えばエディット開始番地を連続的にずらせる ストップキーまたは^Cを押すとエディットメモリーモードを抜ける。HELPキーを押すとエディットメモリーモードで使うキーの説明を表示する。ストップキーまたは、^C以外のキーを押せばHELPキーを押す前の画面を表示する
F	フィル・メモリー	f<開始番地>，<終了番地>，<定数> メモリーの内容を、0から255までの定数でうめる
G	ゴー	g〔<実行開始番地>〕〔，<ブレークポイント番地1>〕 〔，<ブレークポイント番地2>〕 機械語プログラムを実行する。ブレークポイント番地を実行しようとするとその番地を次のように表示して実行を一時停止する *<セグメントベース番地>：<相対番地> 再びGコマンドを指示すればこの番地から実行を再開する
I	インプット	i<ポート・アドレス> 入力ポートの値を画面に表示する
L	ディスアセンブル （リスト）	l〔<開始番地>〕〔，<終了番地>〕 メモリーの内容を逆アセンブルして画面に表示する。プリントモードになっていればプリンターにも出力する　（注1）
M	ムーブ・メモリー	m<転送元先頭番地>，<転送元終了番地>，<転送先先頭番地> メモリー間で値の転送をする
O	アウトプット	o<ポート・アドレス>，<データ> 指定したポート・アドレスにデータを出力する
P	プリンタースイッチ	p プリントモードの設定・解除をする。このコマンドを実行するたびにモードが切り替る
R	リード・テープ	r〔<ファイル名>〕 Wコマンドでカセットテープにセーブしたメモリーの内容をロードする。セーブした時と同じ番地にロードする

コマンド	意　味	入　力　形　式
S	セット・メモリー	s〔＜開始番地＞〕 メモリーの内容を表示し、新しい内容を聞いてくる ssw〔｛1～7｝〕 メモリースイッチの内容を表示し、新しい内容を聞いてくる どちらの場合も、数値を入れずにスペースキーかリターンキーをおせば、内容は変わらない スペースキーを押せば次の番地（スイッチ）の内容を表示し、リターンキーを押せばモニターに戻る メモリーが対象の時に＾Bを押すと前の番地の内容を表示する
TM	テスト・メモリー	tm 実装しているメモリーのテストをする。異常が無ければ Test Complete! と表示する
V	ベリファイ・テープ	v〔＜ファイル名＞〕 Wコマンドでカセットテープにセーブしたデータと実際のメモリーの内容を比較する。内容が一致しなければ”？”を表示する
W	ライト・テープ	w〔＜デバイス名＞〕〔＜ファイル名＞〕，＜開始番地＞，＜セーブ終了番地＞ メモリーの内容をカセットテープにセーブする
X	イグザミンレジスター	x｛〔＜CPUレジスター＞〕または〔＜CPUフラグ名＞〕｝ ＜CPUレジスター＞：AX，BX，CX，DX，SP，BP，SI，DI，CS，DS，SS，ES ＜CPUフラグ名＞：O，D，I，T，S，Z，A，P，C レジスター・フラグの内容を表示し、新しい内容を聞いてくる レジスターが対象の時は、数値を入れずにスペースキーかリターンキーをおせば内容は変わらない フラグが対象の時は、数値を入れずにリターンキーをおせば、内容は変わらない
＾A 【HELP】	ヘルプ	＾a 【HELP】 モニターで使えるコマンドとパラメーターを画面に表示する
＾B	リターン	＾b モニターモードからBASICモードに戻る
＾D	ダンプ・ディスク	＾d＜ドライブ番号＞〔，＜サーフェス番号＞〕，＜トラック番号1＞，＜セクター番号1＞，〔＜トラック番号2＞，＜セクター番号2＞〕 ディスクの内容を画面に表示する、プリントモードになっていればプリンターにも出力する　（注1）
＾R	リード・ディスク	＾r＜ドライブ番号＞〔，＜サーフェス番号＞〕，＜トラック番号＞，＜セクター番号＞，＜開始番地＞，＜終了番地＞ ディスクからメモリーにデータをロードする
＾W	ライト・ディスク	＾w＜ドライブ番号＞〔，＜サーフェス番号＞〕，＜トラック番号＞，＜セクター番号＞，＜開始番地＞，＜終了番地＞ メモリーの内容をディスクにセーブする

(注1) 【STOP】または【CTRL】＋【C】とすれば途中でも表示をやめる。【CTRL】＋【S】とすれば表示を一時停止する。【STOP】または【CTRL】＋【C】以外のキーを押せば表示を再開する。
(注2) ＾（英文字）は【CTRL】と英文字キーを同時に押すことを示す。

16．通信機能

ＰＣの通信機能はターミナルモードで使う方法と、ＢＡＳＩＣモードの通信機能を使う方法がある。ターミナルモードは文字どおり、ＰＣをホストコンピューターの端末装置として接続する場合に使う。ＢＡＳＩＣモードでは、主としてＰＣにＸＹプロッターなどの周辺装置を接続して制御する場合に使う。
ＰＣをホストコンピューターに接続し、ターミナルモードにはないＲＪＥ（遠隔ジョブ入力）などの機能を実現するには、ＢＡＳＩＣモードで接続し、ＰＣ側で制御プログラムを作る必要がある。
ＰＣ－９８０１とＰＣ－８８０１の通信機能はほとんど同じである。そこで本章では、ＰＣ－９８０１について説明し、最後の節でＰＣ－９８０１とＰＣ－８８０１の違いを説明する。

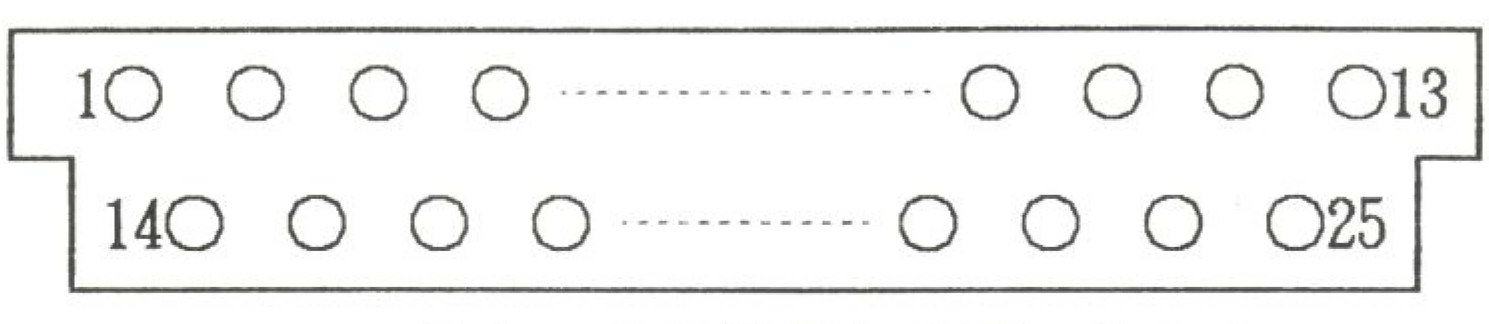
☆ＲＳ－２３２Ｃインタフェース☆

ＲＳ－２３２Ｃは、ＥＩＡ（米国電子工業会）が決めたコンピューターとモデムの間のインタフェースである。ＣＣＩＴＴ（国際電信電話諮問委員会）勧告Ｖ．２４で定義しているインタフェースとほぼ同じ仕様である。現在はモデムとの接続だけでなく、ＸＹプロッターなどシリアル・インタフェースを持つ周辺装置をつなぐインタフェースとして広く普及している。
ＰＣ－９８０１にはＢＡＳＩＣプログラムが動く「ＢＡＳＩＣモード」と、ホストコンピューターの端末として働く「ターミナルモード」がある。どちらのモードでも通信にはＲＳ－２３２Ｃインタフェースを使う。

《信号の位置と種類》

端子番号	信号名	信号の方向	機能
１	ＧＮＤ	←→	保守用アース
２	ＴＸＤ	ＰＣ→	ＰＣが送信するデータ
３	ＲＸＤ	ＰＣ←	ＰＣが受信するデータ
４	ＲＴＳ	ＰＣ→	ＰＣが送信を要求する時にこの信号をＯＮにする
５	ＣＴＳ	ＰＣ←	この信号がＯＮであればＰＣはデータを送信できる
６	ＤＳＲ	ＰＣ←	この信号がＯＮであればＰＣはデータの受信を求められている
７	ＧＮＤ	←→	通信用アース
８	ＤＣＤ	ＰＣ←	この信号がＯＮであればＰＣはデータを受信できる（注）
９	ＮＣ		
１０	ＮＣ		
１１	ＧＮＤ		
１２	ＮＣ		
１３	ＧＮＤ		
１４	ＧＮＤ		
１５	ＴＸＣ(2)	ＰＣ←	内部送信タイミング（同期式のみ）　（注）
１６	ＮＣ		
１７	ＲＸＣ	ＰＣ←	受信タイミング（同期式のみ）　（注）
１８	ＮＣ		
１９	ＮＣ		
２０	ＤＴＲ	ＰＣ→	ＰＣが送受信できる状態であればこの信号をＯＮにする
２１	ＮＣ		
２２	ＮＣ		
２３	ＮＣ		
２４	ＴＸＣ(1)	ＰＣ→	外部送信タイミング（同期式のみ）　（注）
２５	ＮＣ		

（注）　ＤＣＤは主に相手がモデムの時に使う
ＴＸＣ(1)・ＴＸＣ(2)はどちらか一方のみ使う。
ＰＣ－９８０１では非同期式しか使えないので、ＴＸＣ(1)・ＴＸＣ(2)・ＲＸＣは使わない。



ＲＳ－２３２Ｃおす形コネクター

☆信号線の接続方法☆

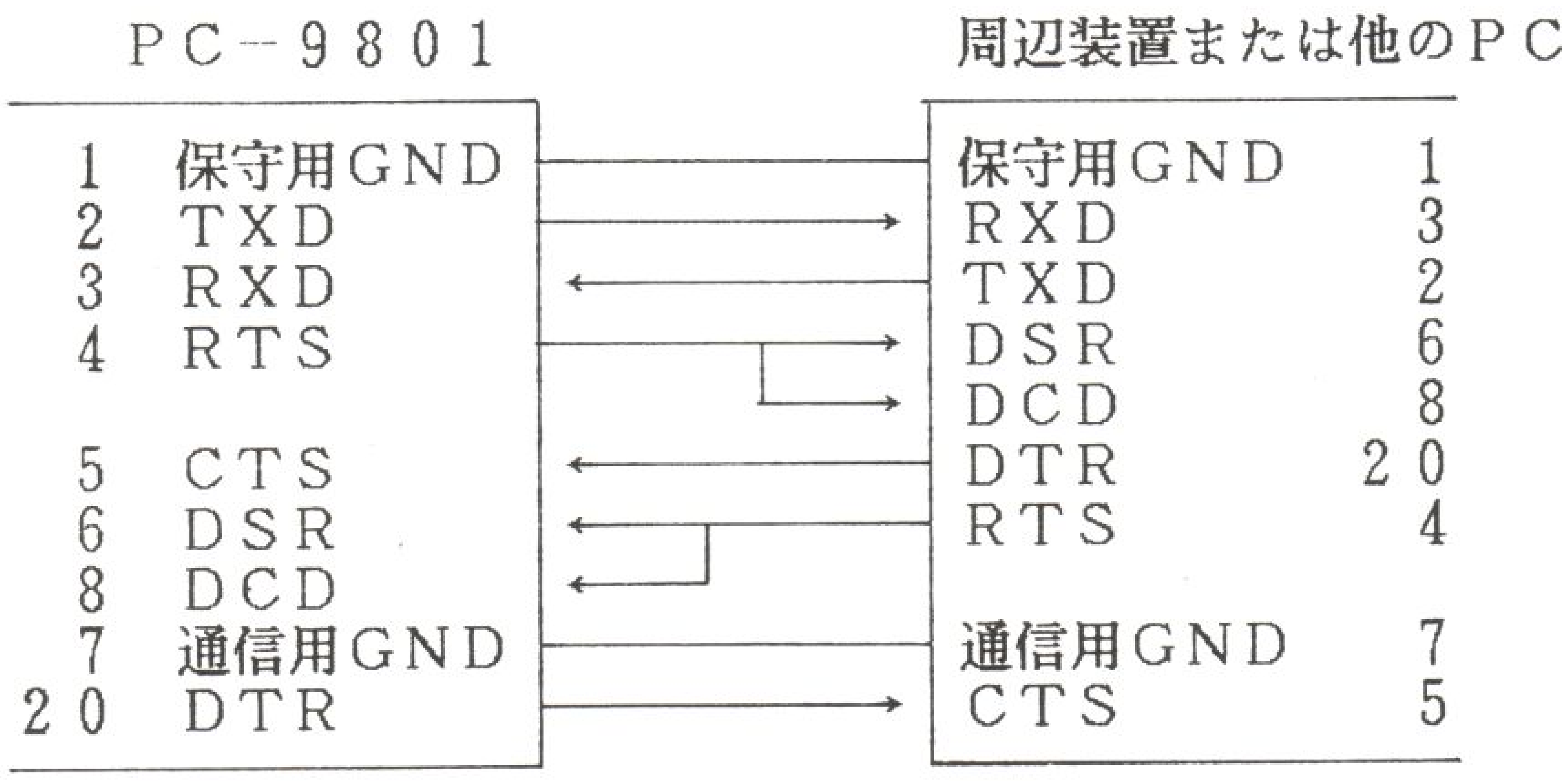
ＲＳ－２３２Ｃインタフェース間の信号線の接続方法は、通信の制御方法によって二種類に大別できる。一つは信号線（ＲＴＳ・ＣＴＳなど）を使ってハードウエア的に制御する方法である。具体的な方法はたくさんあるが、フルハンドシェイクと呼ぶ制御方法が一般的である。もう一つはハードウエア的な制御をせずにデータをたれ流す方法である。この時はソフトウエアで通信の制御をする。

《フルハンドシェイク》
フルハンドシェイクでは、「データを送りたい」と送信側が受信側に知らせる。それに対して受信側が、「送ってもよい」と答えて、初めてデータの送信ができる。実際には４本の信号線を使い、次のような制御をする。
〔ＰＣから周辺装置へデータを送信する手順〕

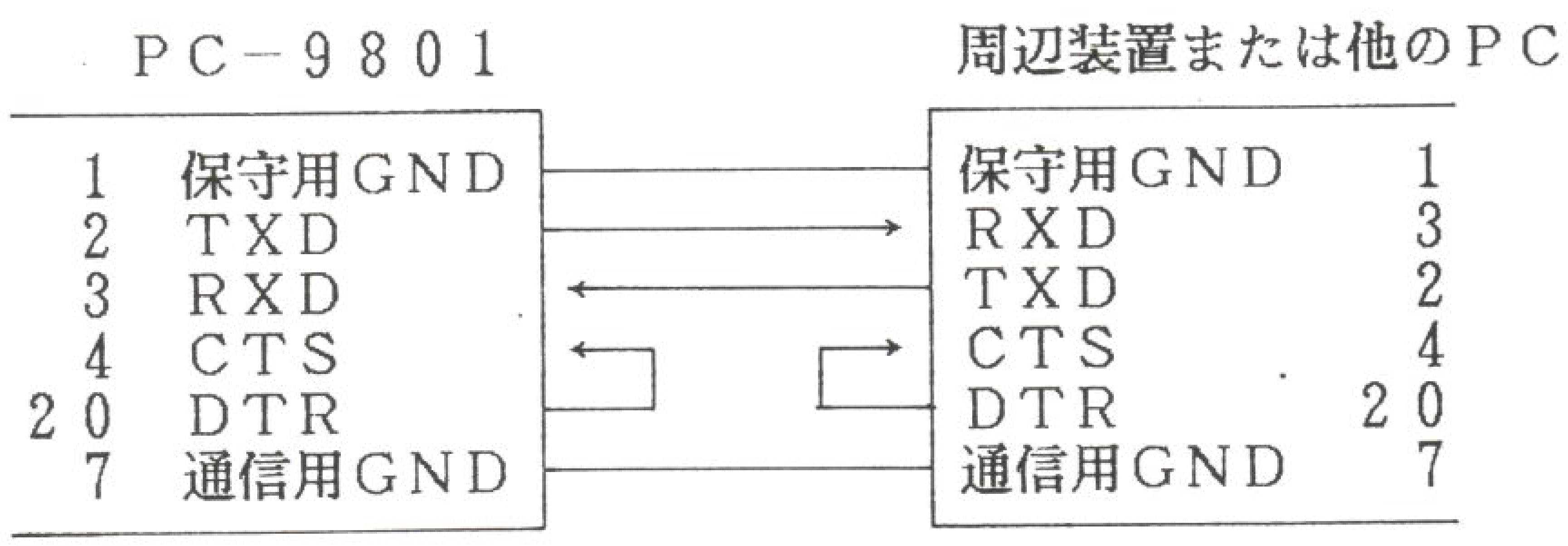
〔ＰＣ－９８０１〕		〔周辺装置〕		
１．ＲＴＳ・ＯＮ	→	ＤＳＲ・ＯＮ	……	ＰＣ－９８０１はＲＴＳ線をＯＮにする。周辺装置のＤＳＲ線もＯＮになる
２．ＣＴＳ・ＯＮ	←	ＤＴＲ・ＯＮ	……	周辺装置はＤＳＲ線がＯＮで受信可能ならＤＴＲ線をＯＮにする。ＰＣ－９８０１のＣＴＳ線もＯＮになる
３．ＴＸＤ・データ	→	ＲＸＤ・データ	……	ＰＣ－９８０１はＣＴＳ線がＯＮであれば、ＴＸＤ線を使ってデータを送信する。周辺装置はＲＸＤ線によってこのデータを受信する
４．ＲＴＳ・ＯＦＦ	→	ＤＳＲ・ＯＦＦ	……	データの送信が終わったらＰＣ－９８０１はＲＴＳ線をＯＦＦにする。周辺装置のＤＳＲ線もＯＦＦになる

（注）周辺装置からＰＣへデータを送信する手順は、この逆である。

フルハンドシェイクでは次のような結線をするのが一般的である。



《たれ流し方式》
送信側がデータを送りたい時にいつでもデータを送ってしまう方式である。ＣＴＳ線がＯＮになっていなければ、送信できない場合が多いので、自分側のＤＴＲ線とＣＴＳ線をショートさせる。ＰＣ－９８０１どうしを接続するときは通常この方式を使う。たれ流し方式では次のような結線をするのが一般的である。



☆通信に必要な規約☆

ＲＳ－２３２Ｃ規格は信号線の意味を決めているだけである。一方、実際の通信では、データビット長・ボーレートなど、さらに細かく仕様を決める必要がある。また、ターミナルモードでは、これに加えてリターンキーの送信処理やＣＲコードの受信処理などについても定めなければならない。

ＰＣ－９８０１では、このような仕様をメモリースイッチやディップスイッチを使って設定する。また、ＴＥＲＭ文やＯＰＥＮ文のパラメーターを使っても設定できる。

《フロー制御》

コンピューターはデータを受信すると、いったん受信用バッファにたくわえる。プログラムが入力命令を実行すると、この受信用バッファからデータを取り出す。入力命令を使ってデータを取り出したあとで複雑な処理をすると、データを取り出す速度がデータを受信する速度よりも遅くなり、受信バッファがあふれてしまう場合がある。受信バッファのあふれを防ぐしくみが「フロー制御」である。

① ＰＣ－９８０１は受信バッファ（最大２５６バイト）にたくわえたデータが３／４以上になると、装置制御コードＤ３（13H）を送信側へ送り、データの送信を中断してもらう
② 送信側はＤ３を受けると、装置制御コードＤ１（11H）がくるまでデータの送信を中断する
③ ＰＣ－９８０１は受信バッファに蓄えたデータが１／４以下になったらＤ１を送信側へ送り、データの送信を再開してもらう
④ 送信側はＤ１を受けたら、データの送信を再開する
（Ｄ３（13H）・Ｄ１（11H）については、「キャラクター・コード表」参照のこと）

ＴＥＲＭ文・ＯＰＥＮ文の＜Ｘパラメーター＞で指定すれば、ＲＳ－２３２Ｃを使った通信でＰＣ－９８０１はフロー制御をする。

ＰＣ－９８０１が受信側の場合、ターミナルモードでも入出力モードでも、通信バッファの状態によって①・③で示した動きをする。しかし、ＰＣ－９８０１が送信側の場合にはモードによって動きが異なる。入出力モードでは、Ｄ３・Ｄ１が送られてきても何もしない。フロー制御をするためには、Ｄ３・Ｄ１を拾い、②・④で示したような動きをするプログラムを作らなければならない。

ターミナルモードでは、Ｄ１を受信した時は④に示した通りに動く。Ｄ３を受信した時の処理は通信方式によって多少異なる。全二重モードでは受信したと同時にキー入力が無効になる。半二重モードではＤ３を受信しても現在入力の行だけは送信し、その後キー入力が無効になる。

《シフトコード制御》

ＰＣ－９８０１は8ビットコード（ＪＩＳ８単位符号）を使っているので、8ビット目が1になっている文字コードを持つ文字（カナ文字など）をそのまま表現できる。一方、パソコンや周辺装置の中にはＡＳＣＩＩコードやＪＩＳ７単位符号などの7ビットコードを使うものがある。このような7ビットコードを使う機器では、カナ文字などの7ビットで表現できない文字を制御符号ＳＯ（０ＥＨ）・ＳＩ（０ＦＨ）を使って表現する。ＰＣ－９８０１とＪＩＳ７単位符号を使っている機器を接続した時は、お互いにカナ文字の扱い方が違うため、コードの変換が必要である。このコードの変換を「シフトコード制御」という。

ＴＥＲＭ文・ＯＰＥＮ文の＜Ｓパラメーター＞で指定すれば、ＰＣ－９８０１はＲＳ－２３２Ｃを使った通信でシフトコード制御をする。

A	B	ア	イ	C	D
41	42	B1	B2	43	44

〔ＪＩＳ８単位符号〕

＜－－－－－－－－－＞ シフトコード制御

A	B	SO	ア	イ	SI	C	D
41	42	0E	41	42	0F	43	44

〔ＪＩＳ７単位符号〕

《日本語シフトコード》

ＰＣ－９８０１では文字列中に漢字を含んでいる場合、その前後に２バイトの制御コードＫＩ・ＫＯを付けている。ＰＣ－９８０１内部では次のようになる。

ＫＩ＝ＥＳＣ＋Ｋ＝＆Ｈ１Ｂ４Ｂ　　ＫＯ＝ＥＳＣ＋Ｈ＝＆Ｈ１Ｂ４８

しかし、他のコンピューターや周辺機器ではＫＩ・ＫＯとして異なるコードを使っている場合がある。

ターミナルモードでは、コードの異なるＫＩ・ＫＯを持つコンピューターとの通信で、ＫＩ・ＫＯコードの自動変換をする。ＴＥＲＭ文の＜日本語シフトコード＞でＰを指定すると無変換で通信し、Ｉを指定するとＫＩ・ＫＯを次のように変換して通信する。

ＫＩ＝ＳＢ＋ｐ＝＆Ｈ１Ａ７０　　ＫＯ＝ＳＢ＋ｑ＝＆Ｈ１Ａ７１

〔通信パラメーター一覧表〕

	通信の仕様	内　　　容	PC-9801 パラメーター		PC-8801 パラメーター	
				スイッチ		スイッチ
入出力モード・ターミナルモード共通の仕様	ボーレート	 ７５ １５０ ３００ ６００ １２００ ２４００ ４８００ ９６００	 — — — — — — — —	SW2〔3,2,1,0〕 ＝０００１ ＝００１０ ＝００１１ ＝０１００ ＝０１０１ ＝０１１０ ＝０１１１ ＝１０００	 — — — — — — — —	 ＪＳＷ＝１ ＪＳＷ＝２ ＪＳＷ＝３ ＪＳＷ＝４ ＪＳＷ＝５ ＪＳＷ＝６ ＪＳＷ＝７ ＪＳＷ＝８
	同期タイマー	 内部同期 外部同期	 — —	DSW1 〔7,8,9,10〕 ＝１０１０ ＝０１０１	 — —	 ＪＳＷ＝Ｉ ＪＳＷ＝Ｅ
	パリティー	偶数パリティーチェック 奇数パリティーチェック パリティーチェックなし	Ｅ Ｏ Ｎ	SW1〔5,4〕＝11 SW1〔5,4〕＝10 SW1〔4〕＝0	Ｅ Ｏ Ｎ	DSW2〔1,2〕＝11 DSW2〔1,2〕＝10 DSW2〔1〕＝0
	データビット長	７ビット ８ビット	７ ８	SW1〔3,2〕＝10 SW1〔3,2〕＝11	７ ８	DSW2〔3〕＝0 DSW2〔3〕＝1
	ストップビット長	１ビット １．５ビット ２ビット	１ ２ ３	SW1〔7,6〕＝01 SW1〔7,6〕＝10 SW1〔7,6〕＝11	１ ２ ３	DSW2〔4〕＝0 —————— DSW2〔4〕＝1
	Ｘパラメーター	フロー制御をする フロー制御をしない	Ｘ Ｎ	SW1〔0〕＝1 SW1〔0〕＝0	Ｘ Ｎ	DSW2〔5〕＝1 DSW2〔5〕＝0
	Ｓパラメーター	シフトコード制御をする シフトコード制御をしない	Ｓ Ｎ	SW2〔7〕＝1 SW2〔7〕＝0	Ｓ Ｎ	DSW1〔5〕＝1 DSW1〔5〕＝0
ターミナルモードの仕様	ＤＥＬコード の受信処理	ＢＳ（08）として扱う ＮＵＬ（00）として扱う	Ｂ Ｎ	SW3〔7〕＝0 SW3〔7〕＝1	— —	DSW1〔6〕＝1 DSW1〔6〕＝0
	リターンキー の送信処理	ＣＲ（0D）を送信 ＣＲ＋ＬＦ（0D0A）を送信	Ｃ Ｌ	SW2〔6〕＝0 SW2〔6〕＝1	— —	—————— ——————
	ＣＲコード の受信処理	ＣＲ単独で復帰改行をする ＣＲで復帰動作のみする	Ｃ Ｌ	SW2〔5〕＝0 SW2〔5〕＝1	— —	—————— ——————
	日本語シフト コード	ＫＩ＝ＥＳＣ＋Ｋ ＫＯ＝ＥＳＣ＋Ｈ ＫＩ＝ＳＢ＋ｐ ＫＯ＝ＳＢ＋ｑ	Ｐ Ｉ	SW2〔4〕＝0 SW2〔4〕＝1	— —	—————— ——————
	通信方式	全二重 半二重	Ｆ Ｈ	SW1〔1〕＝0 SW1〔1〕＝1	Ｆ Ｈ	DSW2〔6〕＝0 DSW2〔6〕＝1
	スタック長	リモートＢＡＳＩＣプロトコルを実行する時に使う利用者スタックの長さ				

（注意）　ＴＥＲＭ文やＯＰＥＮ文のパラメーターを省略するか空白で指定した場合は、メモリースイッチで設定した仕様になる。ただし、スタック長は省略すると１０２４バイトになる。

表中のＳＷ１・ＳＷ２・ＳＷ３はメモリースイッチを示し、ＤＳＷ１・ＤＳＷ２はデップスイッチを示している。ＪＳＷはＰＣ－８８０１のジャンパースイッチを示す。スイッチ欄の１はＯＮ、０はＯＦＦを意味する。ただし、ジャンパースイッチではソケットを差し込む場所の名前が書いてある。

☆ターミナルモード☆

基本的にはターミナルモードのＰＣ－９８０１はテレタイプのような無手順の端末として働く。しかし、ホスト側からスペシャルＥＳＣシーケンスを送れば、種々のローカル処理もできる。加えて、リモートＢＡＳＩＣプロトコルを使えば、あたかもキーボードから直接入力したかのように、ホスト側から送ったＢＡＳＩＣステートメントを実行できる。

ＰＣ－９８０１はＪＩＳ８単位符号を拡張したコードを使っている。ＥＢＣＤＩＣコードを使っているホストコンピューターの端末として使う場合は、ホスト側でコード変換する必要がある。大部分のホストコンピューターでは回線を定義する時にコード変換の指定もできる。

《ターミナルモードにする方法》

１．ディップスイッチ２の２番目をＯＮにする
　電源を入れたとき、あるいはリセットボタンを押した時にターミナルモードで立ち上がる。通信に関する各仕様はメモリスイッチで設定したものになる

２．ＮＥＷ　ＯＮ文を使う
　ＮＥＷ　ＯＮ文でソフト的にディップスイッチ２の２番目をＯＮにする方法である

３．ＴＥＲＭ文を使う
　ＴＥＲＭ文を使えばターミナルモードにするだけでなく、通信に関する各仕様もあわせて設定できる。ＴＥＲＭ文で設定した仕様は、電源を切るかＢＡＳＩＣモードに切り替わるまで有効である

ＴＥＲＭ”〔ＣＯＭ：〔＜パリティー＞〔＜データビット長＞〔＜ストップビット長＞
〔＜Ｘパラメター＞〔＜Ｓパラメター＞〔＜ＤＥＬコードの受信処理＞
〔＜リターンキーの送信処理＞〔＜ＣＲコードの受信処理＞〔＜日本語シフトコード＞
〕〕〕〕〕〕〕〕〕〕”〔，〔＜通信方式＞〕〔，スタック長〕〕

(注)　ボーレートの指定はメモリースイッチ、同期クロックの指定はディップスイッチを使う。

《ターミナルモードからＢＡＳＩＣモードにする方法》

【ＳＨＩＦＴ】＋【ＳＴＯＰ】

(注)　前述の２・３の方法でターミナルモードにした場合は、リセットボタンを押してもよい。

《全二重モードと半二重モード》

ＰＣ－９８０１は実際の回線上は全二重（４線）でつなぐ。しかし、ターミナルモードの時は全二重の端末としても半二重の端末としても働く。この切り替えはＴＥＲＭ文の＜通信方式＞で指定する。全二重モードでキーインしたデータはすぐに送信し、端末側にはそのデータを表示しない。半二重モードではキーインしたデータは端末側に表示するだけで、リターンキーを押すまで送信しない。

《ファンクションキー》

ターミナルモードになった時のファンクションキーｆ・１～ｆ・５の内容はＢＡＳＩＣのＫＥＹ文で指定した内容か、指定していなければシステムで決まっている内容である。ｆ・６～ｆ・１０はターミナルモード専用の内容に変わる。

ターミナルモードではファンクションキーを使って通信方式の他にリテラルモード・画面出力モードの設定をできる。ｆ・６～ｆ・８がモードを切り替えるためのキーである。

キーを押すたびにファンクションキー表示の内容が入れ替わる。表中で表示の欄に◎印のついている方を最初に表示する。ｆ・７はＴＥＲＭ文かメモリースイッチで指定した通信モードの逆を表示する。

ｆ・６～ｆ・８のファンクションキーの表示内容は、「そのキーを押した時の働き」であり、「現在のモード」ではない。例えば、ＨＡＬＦと表示している時の通信モードは全二重である。

リテラルモード：制御コードを受けても文字として表示するだけで、それの意味する処理はしない。ただしリターンコードとラインフィードコードは処理する

画面出力モード：画面への表示をプリンターにも出力する

キー	表　示	機　　　能
f・6	◎LTRL N−LTRL	リテラルモードにする ノンリテラルモードにする
f・7	HALF FULL	半二重モードにする 全二重モードにする
f・8	◎LPTon LPToff	画面出力モードにする 画面出力モードを解除する
f・9	CPY　BF	スクロールアップして画面上から消えたデータをプリンターに出力する （画面出力モードの時に有効）
f・10	LPT　LF	プリンターに1行、行送りさせる（画面出力モードの時に有効）

《スペシャルESCシーケンス》

ターミナルモードのPC−9801はデータを受信すると、通常はそれを画面に表示する。ただし、受信したデータが制御コードで、ターミナルモードの状態がノンリテラルモードであれば、その制御コードが示す働きをする。例えばCTRL-Lを受信すれば画面をクリアし、CTRL-Gを受信すればブザーを鳴らす。

この他にESCコードに引き続く一連の文字列（スペシャルESCシーケンス）を受信しても、それを表示せずに特殊な動きをする。例えばESC @と2文字受信すると、以後ディスプレーに表示するデータはプリンターにも出力する。この他に、スペシャルESCシーケンスを使えばカーソルの位置決め・画面のクリア・キーボードのロックなどができる。

ESC @	次にESC Aを受信するか電源を切るまで、ディスプレーへの出力はプリンターへも出力する
ESC A	ESC @で設定したプリンターへの出力モードを解除する
ESC #	次にESC ”を受信するか電源を切るまで、キー入力を無効にする。キー入力したデータは捨てられる
ESC ”	キー入力を有効にする
ESC !	ホストコンピューターへIDコード（NEC00000010 CR.LF）を送信する
ESC T	現在カーソルのある行をクリアする。カーソルは現在行の先頭へ移る
ESC Y	現在のカーソル位置から、スクロールエリアの最下位行までをクリアする。カーソルの位置は変わらない
ESC *	ディスプレー画面をクリアし、カーソルをホームポジションに表示する。CLRキーを入力したのと同じ状態にする
ESC △	ターミナルの状態を、ターミナルモードになった初期状態に戻す　△：空白
ESC =ij	i、jは文字である。カーソルをテキスト画面上の（列、行）=（ASC（j）−&H20，ASC（i）−&H20）に表示する

《リモートBASICプロトコル》

スペシャルESCシーケンスの一つとして、ESC >BASICステートメントという形がある。この形のデータを受信すると、ターミナルモードのPC−9801は、データ中のBASICステートメント部分を実行する。PRINT文があれば、その結果を自分のディスプレーに表示する。例えばESC >FOR I=1 TO 10:PRINT ”PC−9801”:NEXT Iというデータを受信すると、PC−9801は自分のディスプレーにPC−9801という文字を10回表示する。

このような形のスペシャルESCシーケンスを特に「リモートBASICプロトコル」と呼んでいる。リモートBASICプロトコルは、PRINT文の結果をどこに表示するかによって3つの形がある。

ESC ＞ＢＡＳＩＣステートメント	ＢＡＳＩＣステートメントを実行する。ＰＲＩＮＴ文があれば、実行結果はＰＣ−９８０１の画面に出力する
ESC ＜ＢＡＳＩＣステートメント	ＢＡＳＩＣステートメントを実行する。ＰＲＩＮＴ文があれば、実行結果をホストコンピューターに送信する
ESC ．ＢＡＳＩＣステートメント	ＢＡＳＩＣステートメントを実行する。ＰＲＩＮＴ文があっても実行結果はどこへも表示しないし、送信もしない

グラフィック命令をリモートＢＡＳＩＣプロトコルで実行させた場合には、必ずＰＣ−９８０１のディスプレーに実行結果を表示する。

リモートＢＡＳＩＣプロトコルを使えば、ターミナルモードのＰＣ−９８０１上にプログラムを作ることもできる。例えば、次のようなデータを受信したとする。

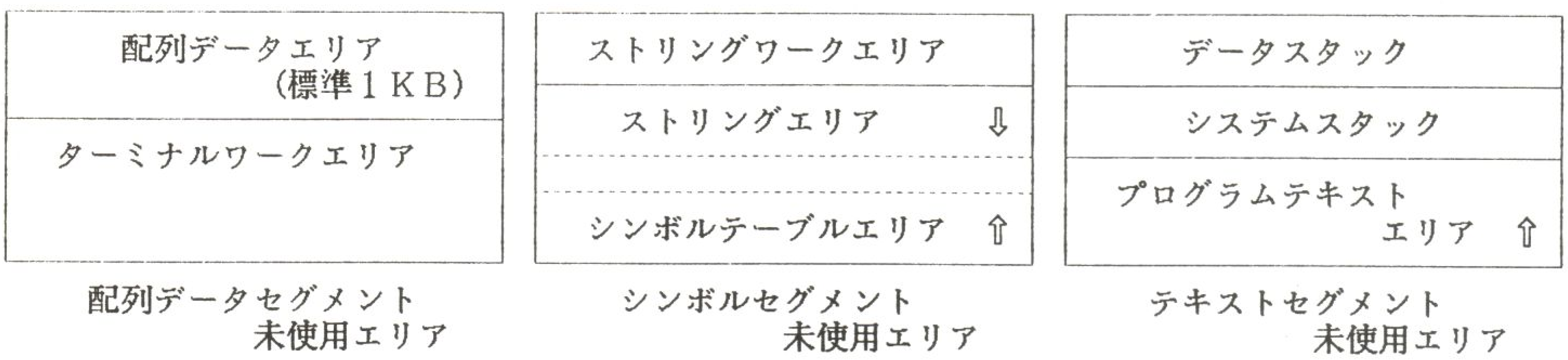
```
ESC ＞１０　FOR　I＝０　TO　１０
ESC ＞２０　PRINT　”ＰＣ−９８０１”
ESC ＞３０　NEXT　I
```

すると、ＰＣ−９８０１には”ＰＣ−９８０１”と１０回表示するプログラムができる。次にESC ＞ＲＵＮというデータを受信すれば、プログラムを実行し画面に１０回”ＰＣ−９８０１”と表示する。ターミナルモードの時にリモートＢＡＳＩＣプロトコルで作るプログラムは、ＢＡＳＩＣモードで作るものとは別の領域にできる。したがって、ターミナルモードを抜けてＢＡＳＩＣモードになると、リモートＢＡＳＩＣプロトコルで作ったプログラムは消えてしまう。逆にＢＡＳＩＣモードで既にプログラムがある状態からターミナルモードになった場合、リモートＢＡＳＩＣプロトコルでプログラムを作っても、ＢＡＳＩＣモードの時のプログラムは影響を受けない。

リモートＢＡＳＩＣプロトコルの使う領域は、ＢＡＳＩＣモードで使わなかった領域である。テキストエリアとして未使用テキストエリアを使い、配列データエリアとして未使用配列データエリアを使う。ＦＲＥ関数でメモリーの未使用領域の大きさを調べる場合や、ＣＬＥＡＲ文でメモリーの使用条件を変える場合はリモートＢＡＳＩＣプロトコルで使う領域を対象にするので注意を要する。

ファイルバッファはターミナルモードとＢＡＳＩＣモードで共用している。ファイルをオープンしたままでターミナルモードにした場合、同じファイル番号を使うとＢＡＳＩＣモードのファイルバッファの内容を変えてしまう。内容を変えたくないときは、クローズしてからターミナルモードにする。

《リモートＢＡＳＩＣプロトコル使用時のメモリーマップ》



☆ＢＡＳＩＣモード☆

ＢＡＳＩＣモードでは、ＲＳ−２３２Ｃインタフェースを１つのファイルとして扱う。ターミナルモードに対比させて、このような使い方を「入出力モード」と呼ぶ。ＲＳ−２３２Ｃインタフェースのファイルディスクリプターは次のようになっている。

”ＣＯＭ：〔＜パリティー＞〔＜データビット長＞〔ストップビット長＞
〔＜Ｘパラメーター＞〔＜Ｓパラメーター＞〕〕〕〕〕”

（注）ボーレートの指定はメモリースイッチ、同期クロックの選択はディップスイッチを使う

ＰＣ−９８０１ではＲＳ−２３２Ｃインタフェースも１つのファイルだから、大部分のファイル入出力命

令を使える。ただし、意味がディスクファイルの場合と多少異なることもある。

〔コミュニケーションファイル用入出力命令一覧〕

入　力　形　式	意　　　味
ＢＬＯＡＤ　＜ファイルディスクリプター＞ 〔，＜ロードアドレス＞〕 〔，Ｒ〕	機械語プログラムを受信する
ＢＳＡＶＥ　＜ファイルディスクリプター＞ ，＜開始番地＞，＜長さ＞	機械語プログラムを送信する
ＣＬＯＳＥ　〔〔＃〕＜ファイル番号＞〕	送受信を終了する
ＥＯＦ　（＜ファイル番号＞）	通信バッファが空であれば真になる
ＩＮＰＵＴ　＃＜ファイル番号＞，＜変数＞ 〔，＜変数＞……〕	通信バッファからデータを持ってくる。通信バッファに必要なだけデータがなければ、データが入ってくるまで待つ。データの区切りはＣＲ　”：”　”，”のいずれか
ＩＮＰＵＴ＄（＜文字数＞ 〔，〔＃〕＜ファイル番号＞）	通信バッファから指定した文字数だけデータを取り出しそれを値として返す。必要なだけデータがなければ、データが入ってくるまで待つ
ＬＩＮＥ　ＩＮＰＵＴ　＃＜ファイル番号＞ ，＜文字変数＞	データの区切りがＣＲだけであることを除き、ＩＮＰＵＴ＃と同じ
ＬＯＡＤ　＜ファイルディスクリプター＞ 〔，Ｒ〕	ＢＡＳＩＣプログラムを受信する
ＬＯＣ　（＜ファイル番号＞）	通信バッファ（入力バッファ）中の文字数を返す
ＬＯＦ　（＜ファイル番号＞）	通信バッファの残りバイト数を返す
ＯＰＥＮ　＜ファイルディスクリプター＞ 〔ＦＯＲ　＜モード＞〕 ＡＳ　〔＃〕＜ファイル番号＞	ファイルディスクリプタにより通信の形式を決めて、送受信できるようにする
ＰＲＩＮＴ　＃＜ファイル番号＞ ，〔＜式＞……〕	データを送信する
ＰＲＩＮＴ　＃＜ファイル番号＞ ＵＳＩＮＧ　＜書式制御文字列＞ ；＜式＞〔｛；｜，｝＜式＞……〕	フォーマット付きで、データを送信する
ＳＡＶＥ　＜ファイルディスクリプター＞ 〔，｛Ａ｜Ｐ｝〕	ＢＡＳＩＣプログラムを送信する
ＷＩＤＴＨ　＜ファイルディスクリプター＞ ，＜サイズ＞	通信バッファの大きさを指定する
ＷＩＤＴＨ　＃＜ファイル番号＞ ，＜サイズ＞	通信バッファの大きさを指定する
ＷＲＩＴＥ　＃＜ファイル番号＞ 〔，＜式＞……〕	データを送信する。不要な空白は省略し、値と値の間はコンマでくぎる。文字列は”で囲む

《割り込み処理》

ＩＮＰＵＴ＃・ＬＩＮＥ　ＩＮＰＵＴ＃文を実行した時、通信バッファに必要なだけデータがなければ、プログラムはデータが入ってくるまで停止して待っている。しかし、プログラムを実行しながらメッセージが送られてきた時だけ、それを表示したいことがある。

このように、ある事象（例えば「メッセージが送られてきた」）が起きた時に、実行中のプログラムを一時中断して、他の処理（メッセージの表示）をすることを「割り込み処理」という。割り込み処理を使わなければ、プログラムの実行中にＲＳ－２３２Ｃからのメッセージを拾うことは難しい。

Ｎ８８－ＢＡＳＩＣは、いろいろな割り込み処理用の命令を持っているが、ＲＳ－２３２Ｃを使った通信に関係するのは次の４つである。

入　力　形　式	意　味
ＯＮ　ＣＯＭ　ＧＯＳＵＢ {<行番号１>｜<ラベル１>} 〔，{<行番号２>｜<ラベル２>}〕 〔，{<行番号３>｜<ラベル３>}〕	割り込み処理サブルーチンの開始行を決める。ＲＳ－２３２Ｃからの入力割り込みが起きれば、ここで指定したサブルーチンが動く
ＣＯＭ　ＯＮ	割り込みを許可する
ＣＯＭ　ＯＦＦ	割り込みを禁止する
ＣＯＭ　ＳＴＯＰ	割り込みを一時保留する。ＣＯＭ　ＯＮ文を実行すると保留していた割り込みが起こる

ＯＮ　ＣＯＭ文で指定したサブルーチンを実行している間は、ＣＯＭ　ＳＴＯＰの状態になる。

例として、通常は時間を表示し、１行分のメッセージを受信したらそれを表示するプログラムを考える。

```
100 OPEN "COM:E71N" AS #1
110 ON COM GOSUB 1000
120 COM ON
200 MSGBUF$="":PRINT CHR$(12):CONSOLE 20,4
300 LOCATE 36,7:PRINT TIME$:GOTO 300
1000 CHRCNT=LOC(1)
1010 IF CHRCNT THEN MSGBUF$=MSGBUF$+INPUT$(CHRCNT,1)
1020 CRPOS=INSTR(MSGBUF$,CHR$(13))
1030 IF CRPOS=0 THEN GOTO 2000
1040 LOCATE 0,23:PRINT CHR$(7)+CHR$(13)
1050 PRINT MID$(MSGBUF$,1,CRPOS-1);
1060 MSGBUF$=MID$(A$,CRPOS+1)
1070 GOTO 1020
2000 LOCATE 36,7
2010 RETURN
```

１００番台の部分で、コミュニケーション・ファイルのオープンと割り込み処理ルーチンの指定、割り込みの初期化をしている。

２００番台の部分が画面の初期化で、３００番の文が時間表示をする部分である。

１０００番台と２０００番台の部分が割り込み処理ルーチンである。割り込みが起こると、バッファの内容をすべて文字変数Ａ＄に加える。その後で、Ａ＄中にリターンコードがあれば、先頭からそこまでを画面に表示する。

☆ＰＣ－８８０１では☆

通信機能については、ほとんどＰＣ－９８０１と同じである。ただし、ターミナルモードに関して４点だけ若干の違いがある。

１．ターミナルモードを設定するディップスイッチ

ディップスイッチ１の２番目を使う。当然ＮＥＷ　ＯＮ文で指定するのも、このスイッチである。

2．ＴＥＲＭ文

ＰＣ－９８０１のＴＥＲＭ文にあった＜ＤＥＬコード受信処理＞・＜リターンキーの送信処理＞、＜ＣＲコードの受信処理＞・＜日本語シフトコード＞のパラメーターがない。

```
TERM”〔COM：〔＜パリティー＞〔＜データビット長＞〔＜ストップビット長＞
        〔＜Xパラメーター＞〔＜Sパラメーター＞〕〕〕〕〕〕〕”
                                 〔，〔＜通信方式＞〕〔，スタック長〕〕
```

3．ファンクションキーの表示内容

ファンクションキーｆ・６～ｆ・１０の機能はまったく同じであるが、ファンクションキー表示の内容が異なる。ＰＣ－９８０１ではｆ・６～ｆ・８を押すと表示内容が入れ替わったが、ＰＣ－８８０１はキーを押しても表示内容は変化しない。

f・6	literal
f・7	half／full
f・8	LPT enable
f・9	copy buffer
f・10	LPT feed

4．リモートＢＡＳＩＣプロトコル

ＰＣ－８８０１ではリモートＢＡＳＩＣプロトコルの使用に制限がある。まず第一にターミナルモードで動いているＰＣ－８８０１のモード上にプログラムを作ることはできない。受信したＢＡＳＩＣステートメントをダイレクトモードで実行するだけである。

ＦＲＥ関数でメモリーの未使用領域を調べたり、ＣＬＥＡＲ文でメモリーの使用条件を変えることはできない。ＥＲＡＳＥ文で配列変数を消去することもできない。さらに、ファイルをオープンしたままターミナルモードになった場合は、同じファイル番号を持つファイルをオープンすることができない。

変数や文字列領域は、どちらもＴＥＲＭ文で長さを指定したスタックを使う。文字列領域の再利用もしないので、文字列の代入を何回もしたり、大きな配列を定義する場合は注意を要する。

17. BASICプログラム集

以下のプログラムはＰＣ－９８０１と８８０１に共通して使える。プログラム４～１１はプログラム１２で使うため、ディスクに保存するときは”ｒｅｉ４”～”ｒｅｉ１１”という名前にしておく必要がある。

《１．シーケンシャルファイルの作成》

シーケンシャルファイル”セイセキ”を作成する。番号・名前・点数を入力、１レコードとする。入力終了は【ＳＴＯＰ】。シーケンシャルファイルを新規作成するときのモードは、ＯＵＴＰＵＴ。ＳＴＯＰキーの割り込みの処理に注意。

```
1000 ON STOP GOSUB *ON.STOP
1010 CLS
1020 OPEN "ｾｲｾｷ" FOR OUTPUT AS #1
1030 *LOOP.1
1040   PRINT
1050   STOP ON
1060     INPUT "ﾊﾞﾝｺﾞｳ";B$
1070     INPUT "ﾅﾏｴ"   ;N$
1080     INPUT "ﾃﾝｽｳ"  ;T
1090   STOP STOP
1100   WRITE#1,B$,N$,T
1110   GOTO *LOOP.1
1120 *END.LOOP.1
1130 CLOSE #1
1140 STOP OFF
1150 END
1160 '
1170 *ON.STOP
1180 PRINT "ｵﾜﾘ"
1190 STOP STOP
1200 RETURN *END.LOOP.1
```

《２．シーケンシャルファイルへの追加》

”セイセキ”ファイルにデータを追加する。追加終了は、【ＳＴＯＰ】。シーケンシャルファイルに追加するときのモードは、ＡＰＰＥＮＤ。

```
1000 ON STOP GOSUB *ON.STOP
1010 CLS
1020 OPEN "ｾｲｾｷ" FOR APPEND AS #1
1030 *LOOP.1
1040   STOP ON
1050     INPUT "ﾊﾞﾝｺﾞｳ";B$
1060     INPUT "ﾅﾏｴ"   ;N$
1070     INPUT "ﾃﾝｽｳ"  ;T
1080     PRINT
1090   STOP STOP
1100   WRITE#1,B$,N$,T
1110   GOTO *LOOP.1
1120 *END.LOOP.1
1130 CLOSE#1
1140 STOP OFF
1150 END
1160 '
1170 *ON.STOP
1180 PRINT "ｵﾜﾘ"
1190 STOP STOP
1200 RETURN *END.LOOP.1
```

《3．シーケンシャルファイルを読む》
”セイセキ”ファイルを読み、画面上に番号・名前・点数を表示する。ファイルの終わりはＥＯＦ（1）で判断。

```
1000 CLS
1010 OPEN "ｾｲｾｷ" FOR INPUT AS #1
1020 WHILE NOT EOF(1)
1030   INPUT#1,B$,N$,T
1040   PRINT   B$,N$,T
1050 WEND
1060 CLOSE #1
1070 END
```

《4．マスターの作成（ランダムファイル）》

ランダムファイル”ウリアゲ゛”を作成する。レコード番号はプログラムで指定し、所属コード・名前・商品Ａ・Ｂ・Ｃの売り上げを入力する。入力終了時は【ＳＴＯＰ】。数値はＭＫＳ＄関数で文字項目に変換し、バッファへ送る。ファイルレイアウトは図のとおり。

3バイト	14バイト	4バイト	4バイト	4バイト	227バイト
所属コード	名前	Ａの売り上げ	Ｂの売り上げ	Ｃの売り上げ	未使用

```
1000 ON STOP GOSUB *ON.STOP
1010 CLS
1020 REC.NO=1
1030 OPEN "ｳﾘｱｹﾞ" AS #1
1040 FIELD#1,3 AS F.CODE$,14 AS F.NAME$
1050 FIELD#1,17 AS DUMMY$,4 AS F.URI.A$,4 AS F.URI.B$,4 AS F.URI.C$
1060 *LOOP.1
1070   PRINT
1080   PRINT USING "ﾚｺｰﾄﾞ ﾊﾞﾝｺﾞｳ #### ﾉ ﾃﾞｰﾀ ｦ ﾆｭｳﾘｮｸ ｼﾃ ｸﾀﾞｻｲ";REC.NO
1090   STOP ON
1100     INPUT "ｼｮｿﾞｸ ｺｰﾄﾞ";I.CODE$
1110     INPUT "ﾅﾏｴ"        ;I.NAME$
1120     INPUT "A ﾉ ｳﾘｱｹﾞ" ;I.URI.A
1130     INPUT "B ﾉ ｳﾘｱｹﾞ" ;I.URI.B
1140     INPUT "C ﾉ ｳﾘｱｹﾞ" ;I.URI.C
1150   STOP STOP
1160   LSET F.CODE$ =     I.CODE$
1170   LSET F.NAME$ =     I.NAME$
1180   LSET F.URI.A$=MKS$(I.URI.A)
1190   LSET F.URI.B$=MKS$(I.URI.B)
1200   LSET F.URI.C$=MKS$(I.URI.C)
1210   PUT#1,REC.NO
1220   REC.NO=REC.NO+1
1230   GOTO *LOOP.1
1240 *END.LOOP.1
1250 CLOSE #1
1260 STOP OFF
1270 END
1280 '
1290 *ON.STOP
1300 PRINT "ｵﾜﾘ"
1310 STOP STOP
1320 RETURN *END.LOOP.1
```

《5．レコードの削除（ランダムファイル）》
”ウリアゲ゛”ファイルから、入力したレコード番号のレコードを削除する。削除したレコードは、所属コードをスペースにする。終了時は【ＳＴＯＰ】。

```
1000 ON STOP GOSUB *ON.STOP
1010 CLS
1020 OPEN "ｳﾘｱｹﾞ" AS #1
1030 FIELD#1,3 AS F.CODE$,14 AS F.NAME$
1040 FIELD#1,17 AS DUMMY$,4 AS F.URI.A$,4 AS F.URI.B$,4 AS F.URI.C$
1050 *LOOP.1
1060   PRINT
1070   STOP ON
1080     INPUT "ｻｸｼﾞｮｽﾙ ﾚｺｰﾄﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ";REC.NO
1090   STOP STOP
1100   IF REC.NO<=0 OR REC.NO>LOF(1) THEN *DATA.ERROR
1110   GET#1,REC.NO
1120   IF F.CODE$="   " THEN *DATA.ERROR
1130   PRINT USING "ﾚｺｰﾄﾞ ﾊﾞﾝｺﾞｳ #### ﾉ ﾚｺｰﾄﾞ ｦ ｻｸｼﾞｮ ｼﾏｼﾀ";REC.NO
1140   PRINT
1150   LSET F.CODE$="   "
1160   PUT#1,REC.NO
1170   GOTO *LOOP.1
1180   '
1190   *DATA.ERROR
1200   PRINT "ｶﾞｲﾄｳ ｽﾙ ﾚｺｰﾄﾞ ﾊ ｱﾘﾏｾﾝ"
1210   GOTO *LOOP.1
1220 *END.LOOP.1
1230 CLOSE #1
1240 STOP OFF
1250 END
1260 '
1270 *ON.STOP
1280 PRINT "ｵﾜﾘ"
1290 STOP STOP
1300 RETURN *END.LOOP.1
```

《6．レコードの追加（ランダムファイル）》
”ウリアゲ゛”ファイルにレコードを追加する。レコード番号は自動的にふる。処理終了時は、【ＳＴＯＰ】。追加するレコードは、ＬＯＦ（１）＋１から。

```
1000 ON STOP GOSUB *ON.STOP
1010 CLS
1020 OPEN "ｳﾘｱｹﾞ" AS #1
1030 FIELD#1,3 AS F.CODE$,14 AS F.NAME$
1040 FIELD#1,17 AS DUMMY$,4 AS F.URI.A$,4 AS F.URI.B$,4 AS F.URI.C$
1050 *LOOP.1
1060   REC.NO=LOF(1)+1
1070   PRINT
1080   PRINT USING "ﾚｺｰﾄﾞ ﾊﾞﾝｺﾞｳ #### ﾉ ﾃﾞｰﾀ ｦ ﾆｭｳﾘｮｸ ｼﾃ ｸﾀﾞｻｲ";REC.NO
1090   STOP ON
1100     INPUT "ｼｮｿﾞｸ ｺｰﾄﾞ";I.CODE$
1110     INPUT "ﾅﾏｴ"        ;I.NAME$
1120     INPUT "A ﾉ ｳﾘｱｹﾞ" ;I.URI.A
1130     INPUT "B ﾉ ｳﾘｱｹﾞ" ;I.URI.B
1140     INPUT "C ﾉ ｳﾘｱｹﾞ" ;I.URI.C
1150   STOP STOP
1160   LSET F.CODE$ =     I.CODE$
1170   LSET F.NAME$ =     I.NAME$
1180   LSET F.URI.A$=MKS$(I.URI.A)
1190   LSET F.URI.B$=MKS$(I.URI.B)
1200   LSET F.URI.C$=MKS$(I.URI.C)
```

```
1210   PUT#1,REC.NO
1220   GOTO *LOOP.1
1230 *END.LOOP.1
1240 CLOSE #1
1250 END
1260 '
1270 *ON.STOP
1280 PRINT "ｵﾜﾘ"
1290 STOP STOP
1300 RETURN *END.LOOP.1
```

《7．レコードの変更（ランダムファイル）》
　”ウリアゲ゛”ファイルのレコードを変更する。変更したいレコードのレコード番号を入力する。変更終了時は、【ＳＴＯＰ】。

```
1000 ON STOP GOSUB *ON.STOP
1010 CLS
1020 OPEN "ｳﾘｱｹﾞ" AS #1
1030 FIELD#1,3 AS F.CODE$,14 AS F.NAME$
1040 FIELD#1,17 AS DUMMY$,4 AS F.URI.A$,4 AS F.URI.B$,4 AS F.URI.C$
1050 *LOOP.1
1060   PRINT
1070   STOP ON
1080     INPUT "ﾍﾝｺｳｽﾙ ﾚｺｰﾄﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ";REC.NO
1090   STOP STOP
1100   IF REC.NO<=0 OR REC.NO>LOF(1) THEN *DATA.ERROR
1110   GET#1,REC.NO
1120   IF F.CODE$="   " THEN *DATA.ERROR
1130   PRINT       "ﾚｺｰﾄﾞNO ｺｰﾄﾞ ﾅﾏｴ              ｳﾘｱｹﾞA       B        C"
1140   PRINT USING "###      "        ;REC.NO;
1150   PRINT USING "& & "             ;F.CODE$;
1160   PRINT USING "&              & ";F.NAME$;
1170   PRINT USING "####### "         ;CVS(F.URI.A$);
1180   PRINT USING "####### "         ;CVS(F.URI.B$);
1190   PRINT USING "#######"          ;CVS(F.URI.C$)
1200   INPUT "ｼｮｿﾞｸ ｺｰﾄﾞ";I.CODE$:LSET F.CODE$ =I.CODE$
1210   INPUT "ﾅﾏｴ"       ;I.NAME$:LSET F.NAME$ =I.NAME$
1220   INPUT "A ﾉ ｳﾘｱｹﾞ" ;I.URI.A:LSET F.URI.A$=MKS$(I.URI.A)
1230   INPUT "B ﾉ ｳﾘｱｹﾞ" ;I.URI.B:LSET F.URI.B$=MKS$(I.URI.B)
1240   INPUT "C ﾉ ｳﾘｱｹﾞ" ;I.URI.C:LSET F.URI.C$=MKS$(I.URI.C)
1250   PUT#1,REC.NO
1260   GOTO *LOOP.1
1270   '
1280   *DATA.ERROR
1290   PRINT "ｶﾞｲﾄｳ ｽﾙ ﾚｺｰﾄﾞ ﾊ ｱﾘﾏｾﾝ"
1300   GOTO *LOOP.1
1310 *END.LOOP.1
1320 CLOSE #1
1330 STOP OFF
1340 END
1350 '
1360 *ON.STOP
1370 PRINT "ｵﾜﾘ"
1380 STOP STOP
1390 RETURN *END.LOOP.1
```

《8. レコードの検索（ランダムファイル）》
”ウリアケ゛”ファイルを読み、指定したレコード番号のレコードを画面に表示する。処理終了時は【STOP】。

```
1000 ON STOP GOSUB *ON.STOP
1010 CLS
1020 OPEN "ｳﾘｱｹﾞ" AS #1
1030 FIELD#1,3 AS F.CODE$,14 AS F.NAME$
1040 FIELD#1,17 AS DUMMY$,4 AS F.URI.A$,4 AS F.URI.B$,4 AS F.URI.C$
1050 *LOOP.1
1060   PRINT
1070   STOP ON
1080     INPUT "ｼﾗﾍﾞﾀｲ ﾚｺｰﾄﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ";REC.NO
1090   STOP STOP
1100   IF REC.NO<=0 OR REC.NO>LOF(1) THEN *DATA.ERROR
1110   GET#1,REC.NO
1120   IF F.CODE$="   " THEN *DATA.ERROR
1130   PRINT        "ﾚｺｰﾄﾞNO ｺｰﾄﾞ ﾅﾏｴ              ｳﾘｱｹﾞA        B        C"
1140   PRINT USING "####    "        ;REC.NO;
1150   PRINT USING "& & "            ;F.CODE$;
1160   PRINT USING "&              & ";F.NAME$;
1170   PRINT USING "####### "        ;CVS(F.URI.A$);
1180   PRINT USING "####### "        ;CVS(F.URI.B$);
1190   PRINT USING "#######"         ;CVS(F.URI.C$)
1200   GOTO *LOOP.1
1210   '
1220   *DATA.ERROR
1230   PRINT "ｶﾞｲﾄｳ ｽﾙ ﾚｺｰﾄﾞ ﾊ ｱﾘﾏｾﾝ"
1240   GOTO *LOOP.1
1250 *END.LOOP.1
1260 CLOSE #1
1270 STOP OFF
1280 END
1290 '
1300 *ON.STOP
1310 PRINT "ｵﾜﾘ"
1320 STOP STOP
1330 RETURN *END.LOOP.1
```

《9. 分　類》
”ウリアケ゛”ファイルを読み、所属コードの小さい順番にならびかえ、シーケンシャルファイル”フ゛ンルイ”を作成する。削除したレコードは”フ゛ンルイ”ファイルには書き込まない。

```
1000 DIM T.CODE$(500),T.REC.NO(500)
1010 CLS
1020 OPEN "ｳﾘｱｹﾞ"                AS #1
1030 OPEN "ﾌﾞﾝﾙｲ" FOR OUTPUT AS #2
1040 FIELD#1,3 AS F.CODE$,14 AS F.NAME$
1050 FIELD#1,17 AS DUMMY$,4 AS F.URI.A$,4 AS F.URI.B$,4 AS F.URI.C$
1060 T.LEN=0
1070 FOR REC.NO=1 TO LOF(1)
1080   GET#1,REC.NO
1090   IF F.CODE$="   " THEN *END.IF.1
1100     T.LEN=T.LEN+1
1110     T.CODE$ (T.LEN)=F.CODE$
1120     T.REC.NO(T.LEN)=REC.NO
1130   *END.IF.1
1140 NEXT REC.NO
1150 FOR J=T.LEN-1 TO 1 STEP -1
1160   FOR K= 1 TO J
1170     IF T.CODE$(K)<=T.CODE$(K+1) THEN *END.IF.2
```

```
1180        SWAP T.CODE$ (K),T.CODE$ (K+1)
1190        SWAP T.REC.NO(K),T.REC.NO(K+1)
1200      *END.IF.2
1210    NEXT K
1220  NEXT J
1230  FOR K=1 TO T.LEN
1240    GET#1,T.REC.NO(K)
1250    WRITE#2,F.CODE$,CVS(F.URI.A$),CVS(F.URI.B$),CVS(F.URI.C$)
1260  NEXT K
1270  CLOSE #1,#2
1280  END
```

《１０．報告書作成》

”ブ゛ンルイ”ファイルを読み、報告書を作る。所属コードで商品Ａ・Ｂ・Ｃの売り上げの小計、所属コードの頭１桁で中計、全体で総計をとる。これは集計プログラムの基本型。所属コードが変わったときの処理に注意。

```
1000 P.FM0$=SPACE$(12)+"& &             #########, #########, #########,"
1010 P.FM1$=SPACE$(12)+"& & ｼｮｳｹｲ       #########, #########, #########,"
1020 P.FM2$=SPACE$(12)+"!    ﾁｭｳｹｲ      #########, #########, #########,"
1030 P.FM3$=SPACE$(12)+" << ｿｳｹｲ >>     #########, #########, #########,"
1040 P.FM4$=SPACE$(10)+STRING$(50,"-")
1050 CLS
1060 OPEN "ﾌﾞﾝﾙｲ" FOR INPUT AS #1
1070 T1.URI.A=0: T1.URI.B=0: T1.URI.C=0
1080 T2.URI.A=0: T2.URI.B=0: T2.URI.C=0
1090 T3.URI.A=0: T3.URI.B=0: T3.URI.C=0
1100 FIRST.FLAG=-1
1110 WHILE NOT EOF(1)
1120   INPUT#1,D.CODE$,D.URI.A,D.URI.B,D.URI.C
1130   IF FIRST.FLAG THEN FIRST.FLAG=0: GOTO *END.IF.1
1140     IF        D.CODE$    <>       OLD.CODE$    THEN GOSUB *KEI1
1150     IF LEFT$(D.CODE$,1)<>LEFT$(OLD.CODE$,1)THEN GOSUB *KEI2
1160   *END.IF.1
1170   OLD.CODE$=D.CODE$
1180   PRINT USING P.FM0$;D.CODE$,D.URI.A,D.URI.B,D.URI.C
1190   T1.URI.A=T1.URI.A+D.URI.A
1200   T1.URI.B=T1.URI.B+D.URI.B
1210   T1.URI.C=T1.URI.C+D.URI.C
1220 WEND
1230 IF NOT FIRST.FLAG THEN GOSUB *KEI1: GOSUB *KEI2: GOSUB *KEI3
1240 CLOSE #1
1250 END
1260 *KEI1  ' ｼｮｳｹｲ ﾉ ﾌﾟﾘﾝﾄ
1270 PRINT USING P.FM1$;OLD.CODE$,T1.URI.A,T1.URI.B,T1.URI.C
1280 PRINT       P.FM4$
1290 T2.URI.A=T2.URI.A+T1.URI.A
1300 T2.URI.B=T2.URI.B+T1.URI.B
1310 T2.URI.C=T2.URI.C+T1.URI.C
1320 T1.URI.A=0: T1.URI.B=0: T1.URI.C=0               :RETURN
1330 *KEI2  ' ﾁｭｳｹｲ ﾉ ﾌﾟﾘﾝﾄ
1340 PRINT USING P.FM2$;LEFT$(OLD.CODE$,1),T2.URI.A,T2.URI.B,T2.URI.C
1350 PRINT       P.FM4$
1360 T3.URI.A=T3.URI.A+T2.URI.A
1370 T3.URI.B=T3.URI.B+T2.URI.B
1380 T3.URI.C=T3.URI.C+T2.URI.C
1390 T2.URI.A=0: T2.URI.B=0: T2.URI.C=0               :RETURN
1400 *KEI3  ' ｿｳｹｲ ﾉ ﾌﾟﾘﾝﾄ
1410 PRINT USING P.FM3$;T3.URI.A,T3.URI.B,T3.URI.C :RETURN
```

《11．円グラフ》

”ウリアケ゛”ファイルを読み、総売り上げ高に対する商品別内訳の円グラフと、所属コードの頭1桁別内訳の円グラフを表示する。

```
1000 DEF FNR(P)=2*3.1416*(P-.25)
1010 DIM T.URI(12),T.TITLE$(12),T.COLOR(12)
1020 SCREEN 0: WIDTH 80,25
1030 CLS
1040 FOR T.P=0 TO 12
1050   T.URI(T.P)=0
1060   READ T.TITLE$(T.P),T.COLOR(T.P)
1070 NEXT T.P
1080 GOSUB *READ.ADD
1090 FOR C=0 TO 7:COLOR=(C,0):NEXT C
1100 CLS 3
1110 FOR C=0 TO 7:COLOR=(C,C):NEXT C
1120 WINDOW(-1,-1)-(1,1)
1130 VIEW(50,50)-(250,150)
1140 T.P.START=0:T.P.END=2
1150 GOSUB *EN.PLOT
1160 VIEW(390,50)-(590,150)
1170 T.P.START=3:T.P.END=12
1180 GOSUB *EN.PLOT
1190 CLOSE #1
1200 SCREEN 0
1210 END
1220 '
1230 DATA A,1,B,2,C,3
1240 '
1250 DATA 1,1
1260 DATA 2,2
1270 DATA 3,3
1280 DATA 4,4
1290 DATA 5,5
1300 DATA 6,6
1310 DATA 7,1
1320 DATA 8,2
1330 DATA 9,3
1340 DATA ｿﾉﾀ,4
1350 ' ﾌｧｲﾙ ﾖﾐｺﾐ･ﾃｰﾌﾞﾙｻｸｾｲ
1360 *READ.ADD
1370 OPEN "ｳﾘｱｹﾞ" AS #1
1380 FIELD#1,3 AS F.CODE$,14 AS F.NAME$
1390 FIELD#1,17 AS DUMMY$,4 AS F.URI.A$,4 AS F.URI.B$,4 AS F.URI.C$
1400 FOR REC.NO=1 TO LOF(1)
1410   GET #1,REC.NO
1420   IF F.CODE$="   " THEN *END.IF.1
1430     URI.A=CVS(F.URI.A$)
1440     URI.B=CVS(F.URI.B$)
1450     URI.C=CVS(F.URI.C$)
1460     T.URI(0)=T.URI(0)+URI.A
1470     T.URI(1)=T.URI(1)+URI.B
1480     T.URI(2)=T.URI(2)+URI.C
1490     T.P=VAL(LEFT$(F.CODE$,1))+2
1500     IF T.P=2 THEN T.P=12
1510     T.URI(T.P)=T.URI(T.P)+URI.A+URI.B+URI.C
1520     T.URI.T   =T.URI.T   +URI.A+URI.B+URI.C
1530   *END.IF.1
1540 NEXT REC.NO
1550 RETURN
```

```
1560 ' ｴﾝｸﾞﾗﾌ ﾋｮｳｼﾞ
1570 *EN.PLOT
1580 CIRCLE(0,0),1,7
1590 LINE(0,0)-(0,-1),7
1600 P=0
1610 FOR T.P=T.P.START TO T.P.END
1620   IF T.URI(T.P)=0 THEN *END.IF.2
1630     DP=T.URI(T.P)/T.URI.T
1640     LINE(0,0)-(COS(FNR(P+DP)),SIN(FNR(P+DP))),7
1650     P.COLOR=T.COLOR(T.P)
1660     PAINT(COS(FNR(P+DP/2))*.7,SIN(FNR(P+DP/2))*.7),P.COLOR,7
1670     LOCATE (VIEW(0)+POINT(2))¥8,(VIEW(1)+POINT(3))¥8
1680     PRINT T.TITLE$(T.P)
1690     P=P+DP
1700   *END.IF.2
1710 NEXT T.P
1720 RETURN
```

《１２．メニュー》

例題４～１１で作成したプログラムを，メニュー画面から選んで実行するプログラム。例題４～１１のプログラムは、”ｒｅｉ４”～”ｒｅｉ１１”の名前でディスクにセーブしておく。このプログラムは、”ｒｅｉ１４”でセーブする。入力した作業番号から、対応するプログラムをＲＵＮコマンドで実行する。各作業プログラムのＥＮＤのかわりにＲＵＮ”ｒｅｉ１２”とすれば、各作業終了後再びメニュープログラムに戻ることができる。

```
1000 ON STOP GOSUB *ON.STOP
1010 *LOOP.1
1020   CLS
1030   LOCATE 20, 2: PRINT "      ***  ｻｷﾞｮｳ ﾒﾆｭｰ   ***        "
1040   LOCATE 20, 5: PRINT "1 ﾏｽﾀｰ ｻｸｾｲ            6 ﾌﾞﾝﾙｲ     "
1050   LOCATE 20, 7: PRINT "2 ﾚｺｰﾄﾞ ｻｸｼﾞｮ          7 ｻｸﾋｮｳ     "
1060   LOCATE 20, 9: PRINT "3 ﾚｺｰﾄﾞ ﾂｲｶ            8 ｸﾞﾗﾌ      "
1070   LOCATE 20,11: PRINT "4 ﾚｺｰﾄﾞ ﾍﾝｺｳ                        "
1080   LOCATE 20,13: PRINT "5 ｹﾝｻｸ              STOP ｻｷﾞｮｳ ｵﾜﾘ"
1090   STOP ON
1100     LOCATE 20,17: INPUT "ｻｷﾞｮｳ ﾊﾞﾝｺﾞｳ ﾊ";N$
1110   STOP STOP
1120   IF N$="1" THEN RUN "rei4"
1130   IF N$="2" THEN RUN "rei5"
1140   IF N$="3" THEN RUN "rei6"
1150   IF N$="4" THEN RUN "rei7"
1160   IF N$="5" THEN RUN "rei8"
1170   IF N$="6" THEN RUN "rei9"
1180   IF N$="7" THEN RUN "rei10"
1190   IF N$="8" THEN RUN "rei11"
1200   GOTO *LOOP.1
1210 *END.LOOP.1
1220 CLS
1230 STOP OFF
1240 END
1250 '
1260 *ON.STOP
1270 STOP STOP
1280 RETURN *END.LOOP.1
```

《１３．ランダムファイルのブロック化（準備）》

ランダムファイルは、１セクター＝２５６バイト単位でしか読み書きできない。１レコードが２５６バイト以下だと、残りが使われない。フロッピーディスクを有効利用するために、複数レコードを１セクターに入れる。これをブロック化という。例題では、ブロック化した”ウリアゲ　Ｂ”ファイルを初期化する。１レコードを３２バイトとして、１セクターに８レコード入れる。ＦＩＥＬＤ文の使い方に注意。

```
1000 CLS
1010 *LOOP.1
1020   INPUT "ﾌｧｲﾙ 'ｳﾘｱｹﾞB' ｦ ｼｮｷｶｼﾏｽ｡ ﾖｲﾃﾞｽｶ (y/n) ";YN$
1030   IF INSTR("Nn",YN$)>0 THEN END
1040   IF INSTR("Yy",YN$)=0 THEN *LOOP.1
1050 *END.LOOP.1
1060 OPEN "ｳﾘｱｹﾞB" AS #1
1070 FOR FLD.NO=0 TO 7
1080   FIELD #1,FLD.NO*32    AS DUMMY$,3 AS F.CODE$
1090   FIELD #1,FLD.NO*32 +3 AS DUMMY$,13 AS F.NAME$
1100   FIELD #1,FLD.NO*32+16 AS DUMMY$, 4 AS F.URI.A$
1110   FIELD #1,FLD.NO*32+20 AS DUMMY$, 4 AS F.URI.B$
1120   FIELD #1,FLD.NO*32+24 AS DUMMY$, 4 AS F.URI.C$
1130   LSET F.CODE$ ="   "
1140   LSET F.NAME$ ="             "
1150   LSET F.URI.A$=MKS$(0)
1160   LSET F.URI.B$=MKS$(0)
1170   LSET F.URI.C$=MKS$(0)
1180 NEXT FLD.NO
1190 FOR REC.NO=1 TO 13
1200   PUT#1,REC.NO
1210 NEXT REC.NO
1220 CLOSE #1
1230 PRINT "ｵﾜﾘ"
1240 END
```

《１４．ブロック化したファイルとキー操作》

初期化した”ウリアゲ　Ｂ”ファイルに対して、レコードの登録・削除・変更・検索をする。コード番号０～１００まで登録できる。目的のレコードのあるレコード番号は、＜コード番号＞￥８＋１。

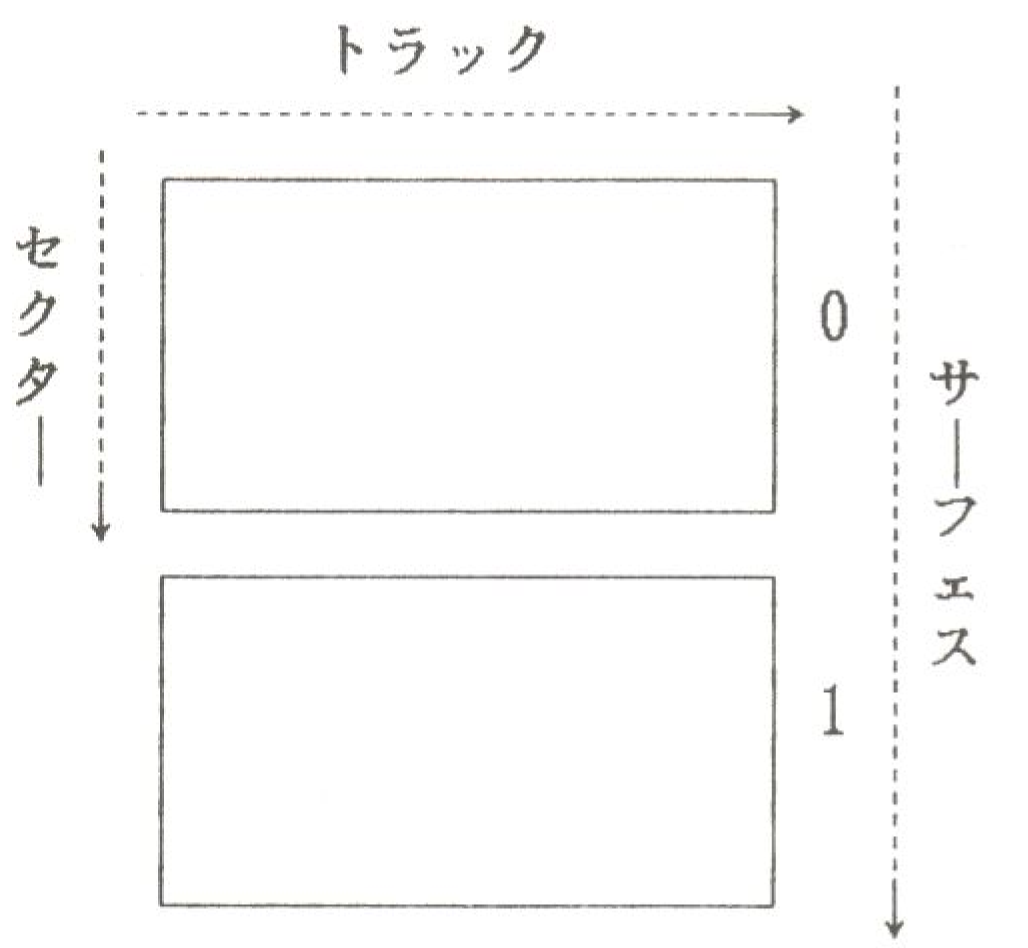
```
1000 OPEN "ｳﾘｱｹﾞB" AS #1
1010 *GET.CODE
1020 ON STOP GOSUB *ON.STOP.PROG
1030 CLS
1040 LOCATE 10,0: PRINT "*** ｳﾘｱｹﾞ ﾌｧｲﾙ ﾉ ｺｳｼﾝ ***"
1050 LOCATE 5,13: PRINT "STOP: ｵﾜﾘ"
1060 STOP ON
1070   GOSUB *U.CODE
1080 STOP STOP
1090 IF I.CODE<0 OR I.CODE>100 THEN *GET.CODE
1100 FIELD #1,(I.CODE MOD 8)*32    AS DUMMY$, 3 AS F.CODE$
1110 FIELD #1,(I.CODE MOD 8)*32+ 3 AS DUMMY$,13 AS F.NAME$
1120 FIELD #1,(I.CODE MOD 8)*32+16 AS DUMMY$, 4 AS F.URI.A$
1130 FIELD #1,(I.CODE MOD 8)*32+20 AS DUMMY$, 4 AS F.URI.B$
1140 FIELD #1,(I.CODE MOD 8)*32+24 AS DUMMY$, 4 AS F.URI.C$
1150 GET#1,I.CODE¥8+1
1160 IF F.CODE$="   " THEN GOSUB *ADD ELSE GOSUB *DISPLAY
1170 *GET.CMND
1180 ON STOP GOSUB *ON.STOP.CMND
1190 FOR I=1 TO 4:LOCATE 5,I*2+3:PRINT USING "   #:";I:NEXT I
1200 LOCATE 5,17:PRINT " RET: ｶｷｺﾑ"
1210 LOCATE 5,19:PRINT " D･d: ｻｸｼﾞｮ"
1220 STOP ON
1230   CMND$=INPUT$(1)
1240 STOP STOP
```

```
1250 IF ASC  (      CMND$)=13 THEN GOSUB *UPDATE:GOTO *GET.CODE
1260 IF INSTR("dD",CMND$)> 0 THEN GOSUB *DEL    :GOTO *GET.CODE
1270 ON INSTR("1234",CMND$) GOSUB *U.NAME,*U.URI.A,*U.URI.B,*U.URI.C
1280 GOTO *GET.CMND
1290 *PROG.END
1300 CLOSE #1
1310 CLS
1320 PRINT "ｵﾜﾘ"
1330 END
1340 '
1350 *ON.STOP.CMND
1360 STOP STOP
1370 RETURN *GET.CODE
1380 ' ﾋｮｳｼﾞ
1390 *DISPLAY
1400 GOSUB *H.CODE : GOSUB *P.CODE
1410 GOSUB *H.NAME : GOSUB *P.NAME
1420 GOSUB *H.URI.A: GOSUB *P.URI.A
1430 GOSUB *H.URI.B: GOSUB *P.URI.B
1440 GOSUB *H.URI.C: GOSUB *P.URI.C
1450 RETURN
1460 ' ﾂｲｶ
1470 *ADD
1480 ON STOP GOSUB *ON.STOP.ADD
1490 ON HELP GOSUB *ON.HELP
1500 LSET F.CODE$=RIGHT$("   "+STR$(I.CODE),3)
1510 GOSUB *H.CODE
1520 GOSUB *P.CODE
1530 LOCATE 5,13: PRINT "STOP: ﾄﾘﾔﾒ"
1540 LOCATE 5,15: PRINT "HELP: ﾓﾄﾞﾙ"
1550 I.NO=1
1560 *DATA.INPUT
1570 STOP ON  : HELP ON
1580   ON I.NO GOSUB *U.NAME,*U.URI.A,*U.URI.B,*U.URI.C
1590 STOP STOP: HELP STOP
1600 I.NO=I.NO+1
1610 IF I.NO<=4 THEN *DATA.INPUT
1620 LOCATE 5,15: PRINT "              "
1630 RETURN
1640 *STOP.INPUT
1650 RETURN *GET.CODE
1660 ' ｶｷｺﾐ
1670 *UPDATE
1680 PUT #1,I.CODE¥8+1
1690 RETURN
1700 ' ｻｸｼﾞｮ
1710 *DEL
1720 LSET F.CODE$="   "
1730 LSET F.NAME$="                    "
1740 LSET F.URI.A$=MKS$(0)
1750 LSET F.URI.B$=MKS$(0)
1760 LSET F.URI.C$=MKS$(0)
1770 PUT #1,I.CODE¥8+1
1780 RETURN
1790 ' ﾃﾞｰﾀ ﾆｭｳﾘｮｸ
1800 *U.CODE  GOSUB *H.CODE : LINE INPUT T$
1810               I.CODE   =      VAL(T$) : RETURN
1820 *U.NAME  GOSUB *H.NAME : LINE INPUT T$
1830          LSET F.NAME$ =          T$   : RETURN
1840 *U.URI.A GOSUB *H.URI.A: LINE INPUT T$
1850          LSET F.URI.A$=MKS$(VAL(T$)): RETURN
```

```
1860 *U.URI.B GOSUB *H.URI.B: LINE INPUT T$
1870          LSET F.URI.B$=MKS$(VAL(T$)): RETURN
1880 *U.URI.C GOSUB *H.URI.C: LINE INPUT T$
1890          LSET F.URI.C$=MKS$(VAL(T$)): RETURN
1900 *U.END   RETURN *PROG.END
1910 *U.HELP  RETURN *DATA.INPUT
1920 *U.STOP  RETURN *STOP.INPUT
1930 '
1940 *ON.STOP.PROG
1950 STOP STOP
1960 RETURN *U.END
1970 '
1980 *ON.STOP.ADD
1990 STOP STOP: HELP STOP
2000 RETURN *U.STOP
2010 '
2020 *ON.HELP
2030 STOP STOP: HELP STOP
2040 IF I.NO=1 THEN RETURN *U.STOP
2050 I.NO=I.NO-1:   RETURN *U.HELP
2060  ' ﾃﾞｰﾀﾒｲ ﾋｮｳｼﾞ
2070 *H.CODE  LOCATE 11, 3: PRINT SPACE$(40)
2080          LOCATE 11, 3: PRINT "ｺｰﾄﾞ --------- ";: RETURN
2090 *H.NAME  LOCATE 11, 5: PRINT SPACE$(40)
2100          LOCATE 11, 5: PRINT "ﾅﾏｴ  --------- ";: RETURN
2110 *H.URI.A LOCATE 11, 7: PRINT SPACE$(40)
2120          LOCATE 11, 7: PRINT "A ﾉ ｳﾘｱｹﾞ --- ";: RETURN
2130 *H.URI.B LOCATE 11, 9: PRINT SPACE$(40)
2140          LOCATE 11, 9: PRINT "B ﾉ ｳﾘｱｹﾞ --- ";: RETURN
2150 *H.URI.C LOCATE 11,11: PRINT SPACE$(40)
2160          LOCATE 11,11: PRINT "C ﾉ ｳﾘｱｹﾞ --- ";: RETURN
2170 '  ﾃﾞｰﾀ ﾋｮｳｼﾞ
2180 *P.CODE  PRINT     F.CODE$   : RETURN
2190 *P.NAME  PRINT     F.NAME$   : RETURN
2200 *P.URI.A PRINT CVS(F.URI.A$): RETURN
2210 *P.URI.B PRINT CVS(F.URI.B$): RETURN
2220 *P.URI.C PRINT CVS(F.URI.C$): RETURN
```

《１５．フロッピーディスクの使用状況を調べる》

フロッピーディスクの使用状況を、グラフィック表示する。表示は次のとおり。



カラー	コード	意　　味
黄色	6	システムの使うクラスター
青	1	未使用クラスター
赤	2	使用中のセクター
緑	4	未使用セクター

使用状況は、フロッピーディスク上のＦＡＴを見る。ＦＡＴの位置は、フロッピーディスクの種類によって異なる。ＤＳＫＦ関数の使い方に注意。ＤＥＦ　ＦＮで定義した関数ＦＮＳＵＦ・ＦＮＴＲＫ・ＦＮＳＥＣはそれぞれ、クラスター番号からサーフェス番号・トラック番号・先頭のクラスター番号を求める。

```
1000 DEFINT A-Z
1010 DEF FNSUF(CLU)=(CLU¥CLU.TRK)MOD(MAX.SUF+1)
1020 DEF FNTRK(CLU)=CLU¥(CLU.TRK*(MAX.SUF+1))
1030 DEF FNSEC(CLU)=(CLU MOD CLU.TRK)*SEC.CLU+1
1040 SCREEN 0: CLS
1050 INPUT "ドライブ NO";DRIVE.NO
1060 MAX.TRK=DSKF(DRIVE.NO,0)
1070 MAX.SEC=DSKF(DRIVE.NO,1)
1080 MAX.SUF=DSKF(DRIVE.NO,2)
1090 CLU.TRK=DSKF(DRIVE.NO,3)
1100 MAX.CLU=DSKF(DRIVE.NO,4)-1
1110 DIR.TRK=DSKF(DRIVE.NO,5)
1120 SEC.CLU=DSKF(DRIVE.NO,6)
1130 FAT.BGN=DSKF(DRIVE.NO,7)
1140 DIM FAT$(MAX.CLU)
1150 CLS 2
1160 SUF1=0: SUF2=MAX.SUF
1170 TRK1=0: TRK2=MAX.TRK
1180 SEC1=1: SEC2=MAX.SEC
1190 BOX.COLOR=1
1200 GOSUB *BOX
1210 FIELD#0, 1 AS FAT$(0)
1220 FOR CLU=1 TO MAX.CLU
1230   FIELD#0,CLU AS DUMMY$, 1 AS FAT$(CLU)
1240 NEXT CLU
1250 IF MAX.TRK=76 THEN DUMMY$=DSKI$(DRIVE.NO,0,DIR.TRK,FAT.BGN)
1260 IF MAX.TRK=39 THEN DUMMY$=DSKI$(DRIVE.NO,1,DIR.TRK,FAT.BGN)
1270 IF MAX.TRK=34 THEN DUMMY$=DSKI$(DRIVE.NO,  DIR.TRK,FAT.BGN)
1280 FOR CLU=0 TO MAX.CLU
1290   FAT=ASC(FAT$(CLU))
1300   IF FAT=&HFF THEN *END.IF.1
1310     SUF=FNSUF(CLU): TRK=FNTRK(CLU): SEC=FNSEC(CLU)
1320     SUF1=SUF: SUF2=SUF
1330     TRK1=TRK: TRK2=TRK
1340     IF FAT=&HFE THEN *ELSE.1.1
1350       IF FAT>&HC0 THEN SEC.USE=FAT AND &H1F ELSE SEC.USE=SEC.CLU
1360       SEC1=SEC: SEC2=SEC+SEC.USE-1
1370       BOX.COLOR=2
1380       GOSUB *BOX
1390       SEC1=SEC+SEC.USE: SEC2=SEC+SEC.CLU-1
1400       BOX.COLOR=4
1410       GOSUB *BOX
1420       GOTO *END.IF.1.1
1430     *ELSE.1.1
1440       SEC1=SEC: SEC2=SEC+SEC.CLU-1
1450       BOX.COLOR=6
1460       GOSUB *BOX
1470     *END.IF.1.1
1480   *END.IF.1
1490 NEXT CLU
1500 END
1510 ' セクター ノ ヒョウジ
1520 *BOX
1530 IF SEC1>SEC2 THEN RETURN
1540 FOR SUF.L=SUF1 TO SUF2
1550   X1= TRK1    *8   : Y1=SUF.L*100+(SEC1-1)*3
1560   X2=(TRK2+1)*8-1: Y2=SUF.L*100+ SEC2    *3-1
1570   LINE(X1,Y1)-(X2,Y2),BOX.COLOR,BF
1580 NEXT SUF.L
1590 RETURN
```

《16．プログラムの自動スタート》
DISK－BASICスタート後、例題14のメニュープログラムを自動スタートさせる。システムディスクにある”setinf.n88”プログラムを使う。アンダーラインで示した箇所は、入力する内容を表す。（98）の場合には次のように操作する。（88）の場合には「フロッピーディスクの使い方」の中の「プログラムの自動スタート」参照。

(1) システムディスクの”setinf.n88”を実行する。

RUN ”setinf.n88”

(2) 同時にオープンするファイルの数を入力する。

```
Set up info sector
Mount a system disk on drive#? 1【RET】
Sure (y/n)?  y【RET】
How many files(0-15)? 2【RET】
```

(3) DISK－BASICスタート後、実行するコマンドを入力する。

```
Text for init?
run ”rei14”【RET】
```

(4) ディスクに、上で指定した情報を書き込む。ドライブ1に書き込むディスクを入れる。

```
# of files to allocate is 2
Text for init is:
run ”rei14”
Mount a system disk on drive# 1
Type RETURN if ready 【RET】

All done.
```

《17．ファイルコピー》
指定した名前のファイルを複写し、新たに指定した名前のファイルを作る。ランダムファイルでもシーケンシャルファイルでもよい。内部形式でセーブしたBASICプログラムや、BSAVEしたファイルは属性が異なるので複写できない。FIELD文に注意。文字変数の最大長は255バイトなので、128バイト×2のフィールドを作る。また、エラー割り込みでファイルの有無をチェックする。

```
1000 ON ERROR GOTO *ERROR.SYORI
1010 CLS
1020 PRINT "ﾌｧｲﾙ ﾒｲ ｦ ｲﾚﾃ ｸﾀﾞｻｲ"
1030 '
1040 *INPUT.I.FILE
1050 INPUT " ｲﾝﾌﾟｯﾄ ";I.FILE$
1060 OPEN I.FILE$ FOR INPUT AS #1: CLOSE #1
1070 '
1080 *INPUT.O.FILE
1090 INPUT " ｱｳﾄﾌﾟｯﾄ";O.FILE$
1100 OPEN O.FILE$ FOR INPUT AS #1: CLOSE#1
1110 PRINT "ｽﾃﾞﾆ ｺﾉ ﾌｧｲﾙｶﾞｱﾘﾏｽ"
1120 INPUT "ｶｻﾈ ﾃ ｶｲﾃ ﾖｲﾃﾞｽｶ (y/n)";YN$
1130 IF YN$<>"y" AND YN$<>"Y" THEN *INPUT.O.FILE
1140 '
1150 *COPY.START
1160 OPEN I.FILE$ AS #1
1170 OPEN O.FILE$ AS #2
1180 FIELD #1,128 AS I.F1$,128 AS I.F2$
1190 FIELD #2,128 AS O.F1$,128 AS O.F2$
1200 FOR REC.NO=1 TO LOF(1)
1210   GET#1
1220   LSET O.F1$=I.F1$
```

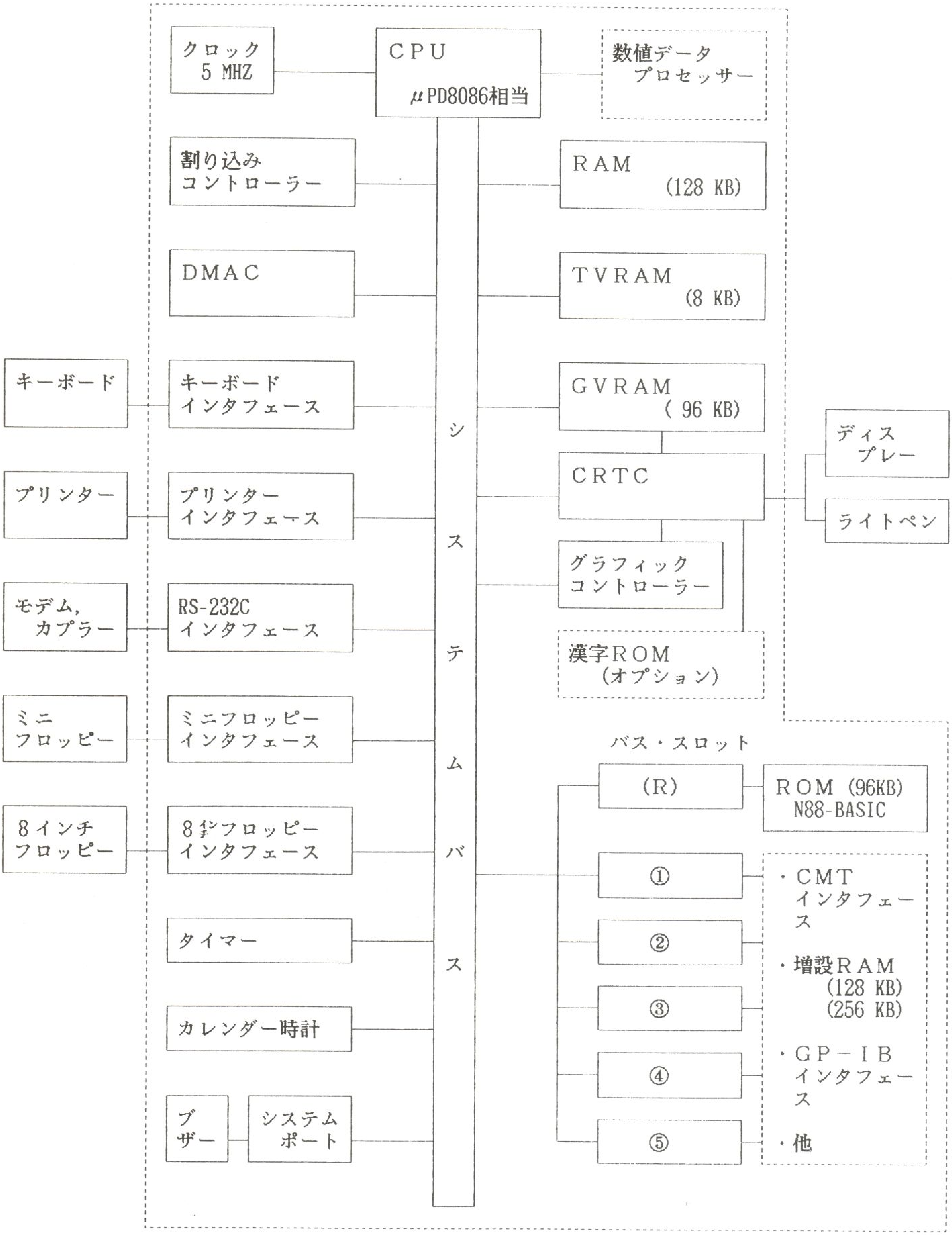
```
1230   LSET O.F2$=I.F2$
1240   PUT#2
1250 NEXT REC.NO
1260 CLOSE #1,#2
1270 PRINT "ｵﾜﾘﾏｼﾀ"
1280 END
1290 '
1300 *ERROR.SYORI
1310 IF NOT(ERL=1060 AND ERR=53) THEN *END.IF.1
1320   PRINT "ｺﾉ ﾌｧｲﾙ ﾊ ｱﾘﾏｾﾝ": RESUME *INPUT.I.FILE
1330 *END.IF.1
1340 IF NOT(ERL=1060           ) THEN *END.IF.2
1350   PRINT "ｲﾚﾅｵｼ"           : RESUME *INPUT.I.FILE
1360 *END.IF.2
1370 IF NOT(ERL=1100 AND ERR=53) THEN *END.IF.3
1380                             RESUME *COPY.START
1390 *END.IF.3
1400 IF NOT(ERL=1100           ) THEN *END.IF.4
1410   PRINT "ｲﾚﾅｵｼ"           : RESUME *INPUT.O.FILE
1420 *END.IF.4
1430 ON ERROR GOTO 0
```

《18．グラフィックで絵を動かす（パレットを変える）》

パレット番号とカラーコードの対応を変えることで絵を動かす。使うパレットカラーは1～7。このうちの1つの表示色を白に、そのほかは黒にすると、画面には白にしたパレットだけを表示する。次々に、白にするパレットを変えることで絵が動く。例題では、ダイヤ型の物体が回転しているように見える。

```
1000 ON STOP GOSUB *ON.STOP
1010 STOP ON
1020 SCREEN 0
1030 CLS 3
1040 GOSUB *SET.COLOR
1050 FOR I=0 TO 48
1060   P.COLOR=(I MOD 7)+1
1070   T=6.28*I/49
1080   X1=COS(T)*200+320: Y1=-SIN(T)*25+100
1090   LINE(X1,Y1)-(320,  0),P.COLOR
1100   LINE(X1,Y1)-(320,199),P.COLOR
1110   LINE(X1,Y1)-(320,100),P.COLOR
1120 NEXT I
1130 P.COLOR=1
1140 *LOOP.1
1150   COLOR=(P.COLOR,0)
1160   IF P.COLOR=7 THEN P.COLOR=1 ELSE P.COLOR=P.COLOR+1
1170   COLOR=(P.COLOR,7)
1180   FOR T=1 TO 20: NEXT T
1190   GOTO *LOOP.1
1200 *PROG.END
1210 GOSUB *SET.COLOR
1220 STOP OFF
1230 END
1240 '
1250 *ON.STOP
1260 RETURN *PROG.END
1270 ' ﾊﾟﾚｯﾄ ｦ ﾓﾄﾞｽ
1280 *SET.COLOR
1290 FOR I=0 TO 7: COLOR=(I,I): NEXT I
1300 RETURN
```

1. 9801 ブロック構成図



2．9801　I/Oアドレスマップ

偶数アドレス
（アドレスのｘは 0,2,4,6,8,A,C,E）

アドレス〔偶数〕	接続機器　〔使用ＬＳＩ〕・偶数アドレスの各ＬＳＩのデータラインはデータバスのD7～D0に接続している
０ｘＨ	割り込みコントローラー　［μPD8259AC相当２個］ ▷タイマー8253C-5 ・キーボード・ＣＲＴ垂直帰線サイクル・ＲＳ－２３２Ｃ・プリンター・８インチフロッピー・数値データプロセッサーからの割り込み要求を管理する　▷周辺機器との入出力のほとんどは、ハードウエアとソフトウエアが密接に関連した割り込みで管理している▷Ｉ／Ｏポートを直接使って入出力する場合は十分注意すること
１ｘＨ	使用禁止（将来、システムの拡張により使う可能性あり）
２ｘＨ	カレンダー時計　［μPD1990AC相当］ ▷カレンダー時計の動作制御と時刻のセットをする　▷現時刻はシステムポートから読み出す　▷ＢＡＳＩＣのＤＡＴＥ＄、ＴＩＭＥ＄などで参照する
３ｘＨ	ＲＳ－２３２Ｃインタフェース　［μPD8251A 相当］ ▷ＲＳ－２３２Ｃインタフェースの動作制御とデータの入出力　▷ＢＡＳＩＣのデバイス名ＣＯＭ：用のポート　▷ターミナルモードのときはホストコンピューターに接続する　▷モデムや音響カプラーとの接続も可能
４ｘＨ	プリンターインタフェース　［μPD8255AC-5相当］ ▷プリンターインタフェースの動作の制御とデータの出力　▷セントロニクス仕様のプリンター PC-8821,PC-8822,PC-8823-C,NK3618-21,NK3618-22などが接続できる
５ｘＨ	メモリーエラーの割り込み制御 ▷標準実装ＲＡＭ、拡張ＲＡＭのパリティーエラーによる割り込みの許可／禁止
６ｘＨ	ＣＲＴコントローラー［μPD7220D-2 相当］（テキスト画面用） ▷ＶＲＡＭのアドレス A0000～A2F9F の内容を表示する
７ｘＨ	ＣＲＴコントローラー　［μPD52611C相当］
８ｘＨ	５インチ固定ディスクインタフェース　▷オプションボード PC-9801-07
９ｘＨ	８インチフロッピーディスクインタフェース　［μPD765AC 相当］ ▷８インチフロッピーディスクインタフェースの動作制御　▷データの入出力はＤＭＡコントローラーを介す　▷フロッピーディスクユニット PC-9881,PC-8881などが接続できる
ＡｘＨ	ＣＲＴコントローラー［μPD7220D 相当］（グラフィック画面用） ▷ＶＲＡＭのアドレス A8000～BFCFF の内容を表示する
ＢｘＨ	使用禁止（将来、システムの拡張により使う可能性あり）
ＣｘＨ	ＯＤＡインタフェース　［μPD8255AC-5相当］▷オプションボード PC-9801-05 ▷ＯＤＡ（アウトプット・デバイス・アダプター）の動作制御とデータの入出力　▷N5243-10,N5233-30,N5233-40などのプリンターが接続できる
ＤｘＨ	使用禁止（将来、システムの拡張により使う可能性あり）
ＥｘＨ	使用禁止（将来、システムの拡張により使う可能性あり）　▷ＢＡＳＩＣのＩＮＰ関数ではキーボードスキャン方式のポートとして扱うが、相当するハードウエアはない　キーボードインタフェースからの信号により、ソフトウエアでシミュレートしている
ＦｘＨ	未使用（Ｆ８～ＦＦは使用禁止）

奇数アドレス
（アドレスのｘは 1,3,5,7,9,B,D,F）

アドレス〔奇数〕	接続機器　［使用ＬＳＩ］ ・奇数アドレスの各ＬＳＩのデータラインはデータバスの D15～D8に接続している
０ｘＨ １ｘＨ	ＤＭＡコントローラー　［μPD8237AC-5相当］ ▷８インチフロッピーディスクとメモリー間のデータを転送する　▷８インチ固定ディスクとメモリー間のデータを転送する　▷ＣＰＵを通さずに直接メモリーと周辺機器の間でデータを転送することをＤＭＡ（ダイレクト・メモリー・アクセス）という　▷ＤＭＡの対象となるアドレス（ＤＭＡアドレス）の下位16ビットはＤＭＡコントローラーにセットする（上位４ビットはＤＭＡバンクにセットする）
２ｘＨ	ＤＭＡバンク ▷ＤＭＡアドレスの上位４ビットを指定する
３ｘＨ	システムポート　［μPD8255AC-5相当］ ▷本体裏ディップスイッチ１の状況・ＲＳ－２３２ＣのCS,CD 信号・カレンダー時計1990ACのDATA OUTなどの読み込み ▷ＲＳ－２３２Ｃからの割り込みの制御・ブザーのオン／オフなど
４ｘＨ	キーボードインタフェース　［μPD8251AC相当］ ▷ダイレクト方式（キーボードマップ参照）
５ｘＨ	５インチフロッピーディスクインタフェース　［μPD8255AC-5相当］ ▷５インチフロッピーディスクインタフェースの動作制御とデータの入出力　▷PC-8001, PC-8801 用ミニフロッピーディスクユニット PC-8031,PC-8031-2W などが接続できる
６ｘＨ	使用禁止（将来、システムの拡張により使う可能性あり）
７ｘＨ	タイマーコントローラー　［μPD8253C-5 相当］ ▷割り込みコントローラー8259へ割り込みを要求するインタバルタイマーとして働く ▷ＢＡＳＩＣのＯＮ　ＴＩＭＥ＄で参照　▷ＲＳ－２３２Ｃの内部同期信号を発生する
８ｘＨ	使用禁止（将来、システムの拡張により使う可能性あり）
９ｘＨ	ＣＭＴインタフェース　［μPD8251AC相当］ ▷オプションボード PC-9801-03 ▷ＣＭＴインタフェースの動作制御とデータの入出力
ＡｘＨ	文字パターンＲＯＭ［μPD23128C相当］ ▷オプションボード PC-9801-01
ＢｘＨ	使用禁止（将来、システムの拡張により使う可能性あり）
ＣｘＨ	ＧＰＩＢ（IEEE-488）インタフェース［μPD7210相当］ ▷オプションボード PC-9801-06 ▷ＧＰＩＢインタフェースの動作制御とデータの入出力
ＤｘＨ	使用禁止（将来、システムの拡張により使う可能性あり）
ＥｘＨ	使用禁止（将来、システムの拡張により使う可能性あり）　▷ＢＡＳＩＣのＩＮＰ関数ではキーボードスキャン方式のポートとして扱うが、相当するハードウエアはない　キーボードインタフェースからの信号により、ソフトウエアでシミュレートしている
ＦｘＨ	未使用　（Ｆ８～ＦＦは使用禁止）

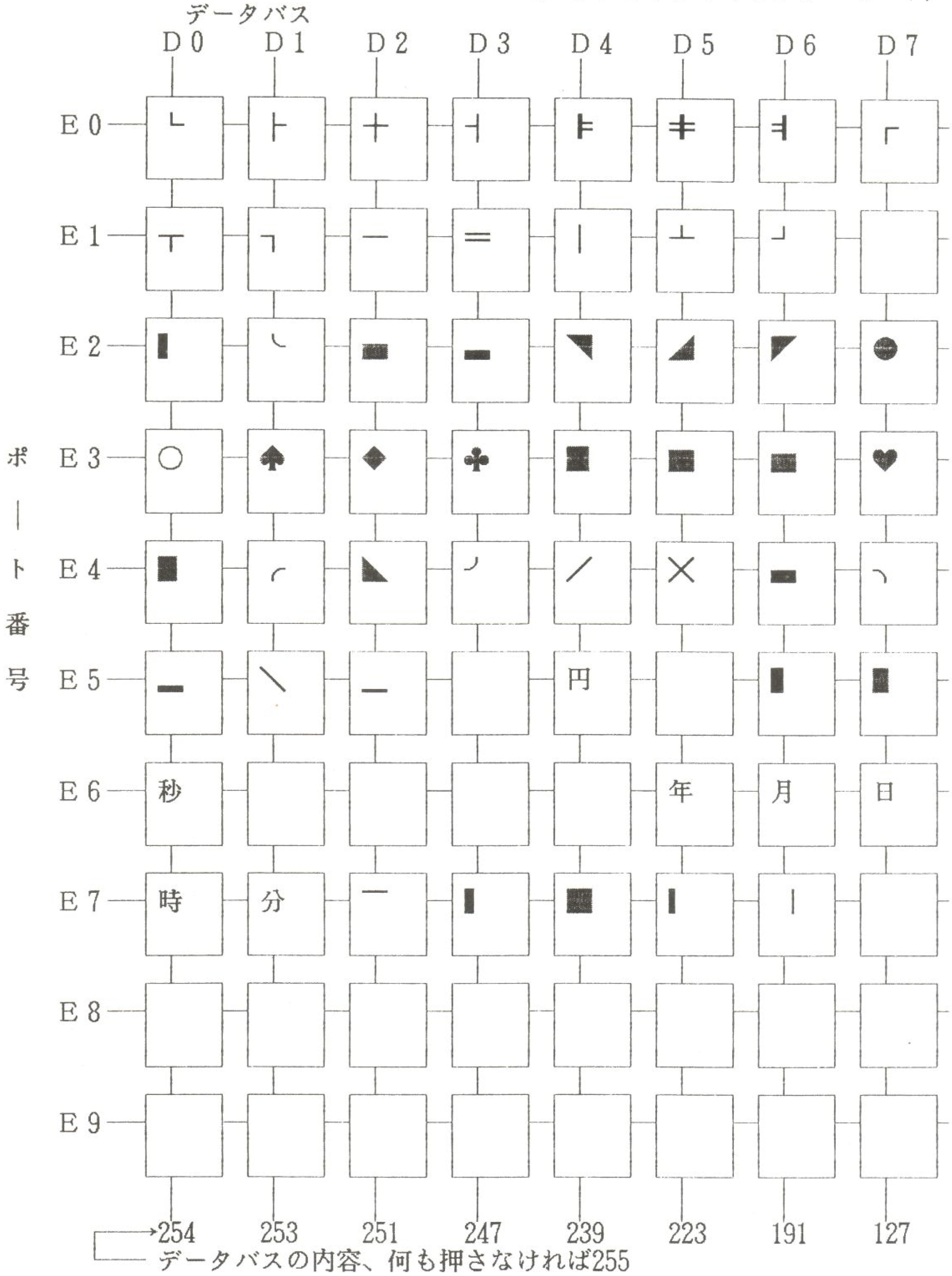
3. 9801 キーボードマップ(キースキャン方式)

データバス

ポート番号	D0	D1	D2	D3	D4	D5	D6	D7
E0	0	1	2	3	4	5	6	7
E1	8	9	*	+	=	,	.	⏎
E2	@ ゛	A チ	B コ	C ソ	D シ	E イ	F ハ	G キ
E3	H ク	I ニ	J マ	K ノ	L リ	M モ	N ミ	O ラ
E4	P セ	Q タ	R ス	S ト	T カ	U ナ	V ヒ	W テ
E5	X サ	Y ン	Z ッ ツ	「 〔 。	¥ ー	」 〕 ム	^ ヘ	= − ホ
E6	ヲ 0 ワ	! 1 ヌ	” 2 フ	# ァ 3 ア	$ ゥ 4 ウ	% ェ 5 エ	& ォ 6 オ	’ ャ 7 ヤ
E7	(ュ 8 ユ	) ョ 9 ヨ	* : ケ	+ ; レ	< 、 , ネ	> 。 . ル	? ・ / メ	_ ロ
E8	HOME CLR	↑	→	DEL	GRPH	カナ	SHIFT	CTRL
E9	STOP	f.1	f.2	f.3	f.4	f.5	SPACE	ESC
EA	TAB	↓	←	HELP	COPY	−	/	CAPS
EB	ROLL UP	ROLL DOWN						
EC	BS	XFER	f.6	f.7	f.8	f.9	f.10	INS
	254	253	251	247	239	223	191	127

→ データバスの内容、何も押さなければ255

（グラフィックキャラクターモード）



4. 9801 キーボードマップ(ダイレクト方式)

ＰＣ－９８０１では、キーボードからのデータをＢＡＳＩＣで直接読み込むことができる。

ＩＮＰ（＆Ｈ４１）命令で読み込まれる入力コードと、キーボードの対応を次に示す。

入力コード		キーボード
注1	注2	
00	80	ESC
01	81	1 ! ヌ
02	82	2 ” フ
03	83	3 # ア ァ
04	84	4 $ ウ ゥ
05	85	5 % エ ェ 年
06	86	6 & オ ォ 月
07	87	7 ’ ヤ ャ 日
08	88	8 (ユ ュ 時
09	89	9) ヨ ョ 分
0A	8A	0 ワ ヲ 秒
0B	8B	－ ＝ ホ ▓
0C	8C	^ ヘ ▓
0D	8D	¥ ー 円
0E	8E	BS
0F	8F	TAB
10	90	Q タ ╭
11	91	W テ ╮
12	92	E イ ィ ◢
13	93	R ス ◣
14	94	T カ ╱
15	95	Y ン ╲
16	96	U ナ ╳
17	97	I ニ ♠
18	98	O ラ ♥
19	99	P セ ▊
1A	9A	@ ゛ ▌
1B	9B	〔 ゜ 「
1C	9C	⏎
1D	9D	A チ ╰
1E	9E	S ト ╯
1F	9F	D シ ◥
20	A0	F ハ ◤
21	A1	G キ ●
22	A2	H ク ○
23	A3	J マ ◆
24	A4	K ノ ♣
25	A5	L リ ■
26	A6	; + レ ┗
27	A7	: * ケ ▁
28	A8	〕 ム 」
29	A9	Z ツ ッ ▁
2A	AA	X サ ▂

入力コード		キーボード
注1	注2	
2B	AB	C ソ ▂
2C	AC	V ヒ ▃
2D	AD	B コ ▄
2E	AE	N ミ ▅
2F	AF	M モ ▆
30	B0	, < ネ 、 ▇
31	B1	. > ル 。 ▏
32	B2	／ ? メ ・ ▕
33	B3	＿ ロ
34	B4	SPACE
35	B5	XFER
36	B6	ROLL UP
37	B7	ROLL DOWN
38	B8	INS
39	B9	DEL
3A	BA	↑
3B	BB	←
3C	BC	→
3D	BD	↓
3E	BE	HOME CLR
3F	BF	HELP
40	C0	－
41	C1	／
42	C2	7 ┏
43	C3	8 ┳
44	C4	9 ┓
45	C5	* ─
46	C6	4 ╞
47	C7	5 ╪
48	C8	6 ╡
49	C9	+ ═
4A	CA	1 ├
4B	CB	2 ┼
4C	CC	3 ┤
4D	CD	= └
4E	CE	0 ┴
4F	CF	, ┘
50	D0	.
51	D1	
52	D2	
53	D3	
54	D4	
55	D5	

入力コード		キーボード
注1	注2	
56	D5	
57	D7	
58	D8	
59	D9	
5A	DA	
5B	DB	
5C	DC	
5D	DD	
5E	DE	
5F	DF	
60	E0	STOP
61	E1	COPY
62	E2	F.1
63	E3	F.2
64	E4	F.3
65	E5	F.4
66	E6	F.5
67	E7	F.6
68	E8	F.7
69	E9	F.8
6A	EA	F.9
6B	EB	F.10
6C	EC	
6D	ED	
6E	EE	
6F	EF	
70	F0	SHIFT
71	F1	CAPS
72	F2	(カナ)
73	F3	GRPH
74	F4	CTRL
75	F5	
76	F6	
77	F7	
78	F8	
79	F9	
7A	FA	
7B	FB	
7C	FC	
7D	FD	
7E	FE	
7F	FF	

注１：キーボードを押している状態で読み込まれるコード
注２：キーボードを押した直後で読み込まれるコード

```
10 A=INP(&H41)
20 PRINT HEX$(A)
30 GOTO 10
```

で確認できる

5. 9801　メモリーアドレスマップ

☆アドレス空間☆

ＰＣ－９８０１は、８０８６相当のＣＰＵを使用している。アドレス空間は、１ＭＢあり、６４ＫＢずつ１６のバンクに分けて管理している。

物理アドレス

物理アドレス	バンク	用途
ＦＦＦＦＦ		
Ｆ００００	バンク　１５	Ｎ８８－ＢＡＳＩＣ インタープリターＲＯＭ（９６ＫＢ）
Ｅ００００	バンク　１４	
Ｄ００００	バンク　１３	システム予備（１６０ＫＢ）
Ｃ００００	バンク　１２	
Ｂ００００	バンク　１１	ＶＲＡＭ（１０４ＫＢ） ・グラフィックＶＲＡＭ -- ９６ＫＢ ・テキストＶＲＡＭ -------- ８ＫＢ （漢字オプションがつくときは４ＫＢふえる） ・不揮発性メモリー -------- ８Ｂ
Ａ００００	バンク　１０	
９００００	バンク　９	増設ＲＡＭ １２８ＫＢ×４＝５１２ＫＢ ２バンク単位で、下位のアドレスから増設可能
８００００	バンク　８	
７００００	バンク　７	
６００００	バンク　６	
５００００	バンク　５	
４００００	バンク　４	
３００００	バンク　３	
２００００	バンク　２	
１００００	バンク　１	標準実装ＲＡＭ（１２８ＫＢ） Ｎ－ＢＡＳＩＣ（８６）で使用するＲＡＭ
０００００	バンク　０	

バンク０～バンク９：Ｎ８８－ＢＡＳＩＣ（８６）
Ｎ８８－ＤＩＳＫ　ＢＡＳＩＣ（８６）で使用するＲＡＭ

☆N８８－ＢＡＳＩＣシステムのメモリーアドレスマップ☆

Ｎ８８－ＢＡＳＩＣ、Ｎ８８－ＤＩＳＫ　ＢＡＳＩＣ（８６）は、増設ＲＡＭの有無に合わせて、メモリー空間を有効に利用する。
メモリー利用の方向は、⇩：上位番地から下位番地へ、⇧：下位番地から上位番地へ。

セグメント	エリア	方向	境界
			←ＨＨ：使用メモリーの容量 （増設ＲＡＭ最大時 ----- A0000） （標準実装ＲＡＭ時 ----- 20000）
機械語 プログラム セグメント	機械語プログラム	⇧	
			←ＭＬ：機械語プログラムの下限番地 （ＣＬＥＡＲ文第２パラメーターで変更可）
配列データ セグメント （注１）	未使用エリア （ＦＲＥ（３）関数で確認可）		
	配列データ	⇧⇧	
			←ＡＬ：配列データセグメントの下限番地
シンボル セグメント	ストリング　ワークエリア （２ＫＢ）	⇩	
	ストリング　格納エリア	⇩⇩	
	未使用エリア （ＦＲＥ（０）関数で確認可）		
	シンボル　テーブルエリア	⇧⇧⇧	
	ワークエリア （２５６ＫＢ）		
			←ＳＬ：シンボルセグメントの下限番地 （ N88-BASIC (86) -------- 10000） （ N88-DISK BASIC (86) -- 16000）
ディスクコード（２４ＫＢ） N88-DISK BASIC (86) のときだけ入る			
			←ＤＬ：ディスクコードの下限番地 （ 10000）
テキスト セグメント	データ　スタック	⇩⇩	
	システム　スタック	⇩⇩	
	未使用エリア （ＦＲＥ（１）関数で確認可）		
	テキスト　エリア	⇧⇧⇧	
	Ｉ／Ｏファイル　バッファ File数×（２５６＋４０）		
	ワークエリア ←（セグメントベースポインター）		
			←ＴＬ：テキストセグメントの下限番地 （ 00600）
システム　エリア			
			←ＬＡ：メモリーの最下限番地　（ 00000）

（注１）
ＣＬＥＡＲ文第三パラメーターでサイズ設定ができる

（注２）
ＡＬ・ＭＬ・ＨＨは、増設ＲＡＭの有無で変わる。各セグメントの下限番地は、セグメントベースポインターに格納されている。ポインターアドレスは、00600 からの相対番地。
下の表を見よ

（注２）
セグメントベースポインター

ポインターアドレス	格納アドレス
１４０２	ＨＨ
１４０Ｅ	ＴＬ
１４１０	ＳＬ
０６Ａ２	ＡＬ
１４０４	ＭＬ

6．9801　ビデオRAM

ビデオＲＡＭには画面に表示する文字やグラフィックに関する情報がはいる。ビデオＲＡＭは，テキスト用とグラフィック用とに分かれる。

☆テキスト用ビデオＲＡＭ（ＴＶＲＡＭ）☆

テキスト用ビデオＲＡＭは、テキスト画面に対応する表示エリアと画面上の文字の属性（色、機能）を示すアトリビュートエリアからなる。テキスト画面には文字と漢字が表示できる。

《文字表示》

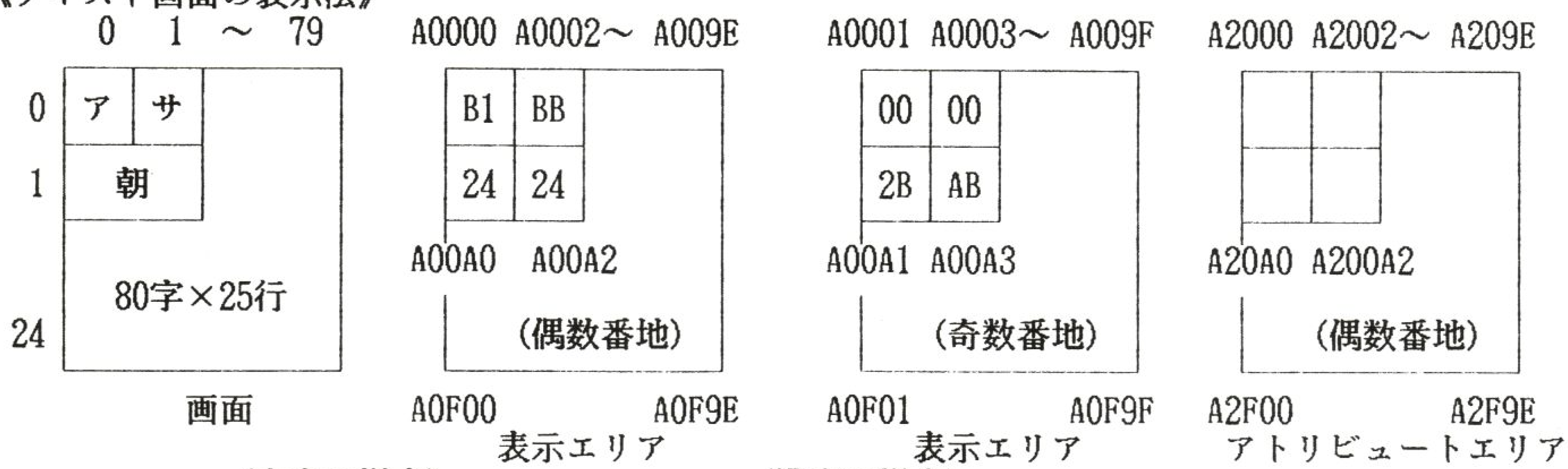
８０字×２５行（２０００字）の画面が、表示エリアのＡ００００Ｈ番地からＡ０Ｆ９ＦＨ番地の４ＫＢに対応する。画面の１文字に対し２バイト割り当てる。この２バイトの上位バイト（偶数番地）に文字表示に必要な文字コードをいれる。下位バイト（奇数番地）は使わない。４０文字モードのときは４の倍数の番地を使う。

《漢字表示》

テキスト画面の２文字分の領域を使って漢字１文字を表示する。

漢字を表示するときは奇数番地も使う。ただし、漢字ＲＯＭボードが必要。漢字１文字当り４バイト割り当て、左半分を２バイト、右半分を２バイトで表す。

《テキスト画面の表示法》



番地	(文字の場合) b7～b0	(漢字の場合) b7	b6	b5～b0
Ａ１（偶数番地）	文字コード	×	×	漢字コード第１バイトから20Ｈ引いたもの
Ａ１＋１（奇数番地）	００　～　００	0	漢字コード第２バイト	
Ａ１＋２（偶数番地）	文字コード	×	×	漢字コード第１バイトから20Ｈ引いたもの
Ａ１＋３（奇数番地）	００　～　００	1	漢字コード第２バイト	

漢字を表示する場合、奇数番地のｂ７は漢字の左半分を表すときは０、右半分を表すときは１となる。×は０でも１でもよい。

（例）

『アサ』と『朝』を画面に表示する。『アサ』は表示エリアの偶数番地に文字コードがはいる。『朝』のＪＩＳ漢字コードは４４２ＢＨだから偶数番地から始まる連続した４バイトに２４Ｈ、２ＢＨ、２４Ｈ、ＡＢＨをいれる。

《アトリビュートエリア》

アトリビュートエリアはＡ２０００Ｈ番地からの４ＫＢ。このうちの偶数番地だけ使い、対応するテキスト画面の文字の色や機能を８ビット（下図のｂ７～ｂ０）で指定する。色指定は、PC-8801 のビデオＲＡＭと同じ。

ビット値	b7	b6	b5	b4	b3	b2	b1	b0
0	黒	黒	黒	文字	ノーマル	ノーマル	ノーマル	シークレット
1	緑	赤	青	たて棒／簡易グラフ	アンダーライン	リバース	ブリンク	ノーマル

《簡易グラフパターン》

１文字の字型を８分割して、８ビットのビットパターンで表した文字を簡易グラフパターンという。簡易グラフパターンを画面に表示するには、表示エリアにビットパターンをセットして対応するアトリビュート

エリアの番地のｂ４を１にして、ポート６８Ｈに１を出力する。ｂ４を１にして、ポート６８Ｈに０を出力するとビットパターンを文字コードとした文字とたて棒『｜』の合成文字ができる。通常はｂ４＝０として表示エリアの文字を表示する。

（表示エリアのビットパターン）

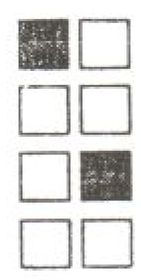
c7	c6	c5	c4	c3	c2	c1	c0

簡易グラフパターン

c0	c4
c1	c5
c2	c6
c3	c7

（例）
『Ａ』の文字コード４１Ｈをビットパターンとすると下の簡易グラフパターンになる

０１０００００１
c7c6c5c4c3c2c1c0



（アトリビュートエリアとポート６８Ｈの設定）

b4 ＝0
OUT &H68,0

対応する文字
（『キャラクターコード表』参照）

（例） Ａ

b4 ＝1
OUT &H68,0

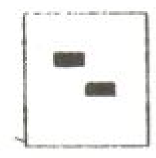
対応する文字とたて棒の合成文字

（例） 

『Ａ』と『｜』の合成

b4 ＝1
OUT &H68,1

簡易グラフパターン

（例） 

ＯＵＴ　＆Ｈ６８，１とした後ＣＯＮＳＯＬＥ　１，２０として１行目がスクロールしないようにしてから、Ａ００００Ｈ番地を６９Ｈ、Ａ０００２Ｈを９６Ｈ、Ａ０００１ＨとＡ０００３Ｈを００Ｈとし、Ａ２０００ＨとＡ２００２ＨをＤ１Ｈにすると黄色で▀▄▀を表示する。

☆グラフィック用ビデオＲＡＭ（ＧＶＲＡＭ）☆

高分解能：０ ～ ６３９、640×400ドット、３９９

		GVRAM0（ビット０）		GVRAM1（ビット１）		GVRAM2（ビット２）		
０ ～ ６３９ 640×200ドット １９９		A8000	A804F	B0000	B004F	B8000	B804F	ＰＡ
		ABE30	ABE7F	B3E30	B3E7F	BBE30	BBE7F	
０ ～ ６３９ 640×200ドット １９９		ABE80	ABECF	B3E80	B3ECF	BBE80	BBECF	ＰＢ
		AFCB0	AFCFF	B7CB0	B7CFF	BFCB0	BFCFF	

グラフィック用ビデオＲＡＭは、上図の３組のＧＶＲＡＭからできている。ＳＣＲＥＥＮ文の第一パラメーターの値によって割り当て方法がかわる。

パレット番号	b2	b1	b0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

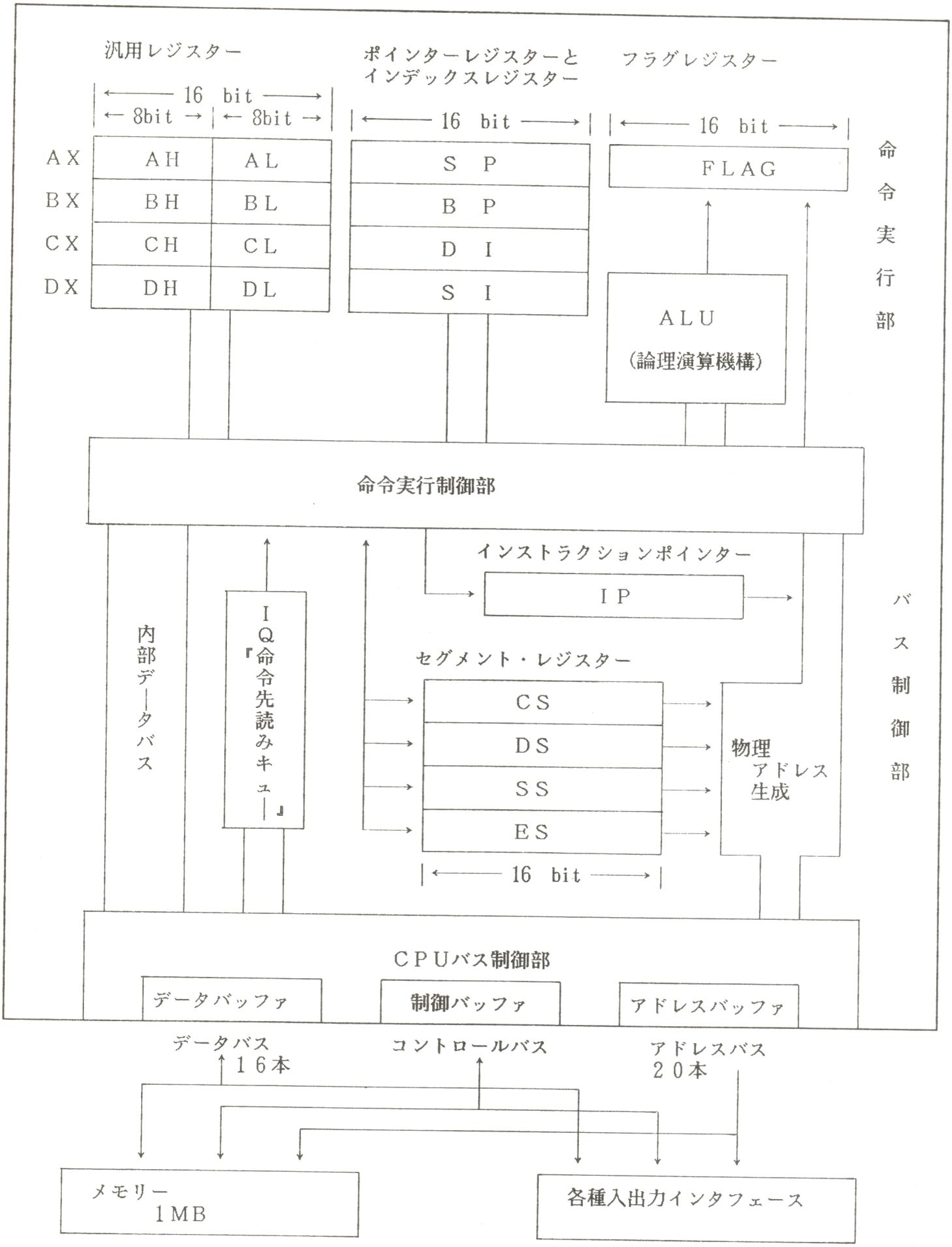
《カラーモード（ＳＣＲＥＥＮの第一パラメーターが０）》
ＧＶＲＡＭ０、１、２をそれぞれビット０、ビット１、ビット２に対応させて各ドットのパレット番号とし、６４０×２００ドットのカラー画面２ページ（ＰＡ、ＰＢ）を表示する。２画面を同時に表示できない。パレット番号は左表のとおり。

《白黒モード（ＳＣＲＥＥＮの第一パラメーターが１）》
ＧＶＲＡＭ０、１、２をさらにＰＡ、ＰＢにわけて６ページの独立した６４０×２００ドットの白黒画面をつくる。ＰＡ、ＰＢそれぞれのなかの３ページは、自由に組み合わせて表示できる。

《高分解能白黒モード（ＳＣＲＥＥＮの第一パラメーターが２）》
専用高解像度ディスプレーで使用する。ＰＡ、ＰＢをひとつにしＧＶＲＡＭ０、１、２がそれぞれ１ページに対応した６４０×４００ドットの画面３ページを構成する。

《高分解能カラーモード（ＳＣＲＥＥＮの第一パラメーターが３）》
専用高解像度ディスプレーで使用する。ＰＡ、ＰＢを一つにして６４０×４００ドットのカラー画面１ページを構成する。

7. 9801 CPU基本ブロック図



☆８０８６の特徴☆

マイクロプロセッサー８０８６は１６ビットＣＰＵである。データを伝達する信号線（データバス）が１６本あり、１６ビット（２バイト）単位でデータを取り扱う。レジスターは１６ビットのものが１４個内蔵されている。その内訳は、汎用レジスターが４個、ポインターとインデックスレジスターが４個、それにセグメントレジスターが４個ある。残りの２個はインストラクションポインターとフラグレジスターである。
接続できるメモリー容量は最大１ＭＢ、アドレスを伝達する信号線（アドレスバス）は２０本ある。メモリーを６５，５３６バイト（６４ＫＢ）単位のセグメントに分割し、セグメントレジスターを使ってプログラムやデータの記憶場所を設定する。
基本命令は約１３５種類あり、ワード（２バイト）単位でデータを操作する命令のほかに、バイト単位でデータを操作する命令もある。８０８０などの８ビットマイクロプロセッサーのプログラムの移植が容易な命令体系になっている。

☆８０８６のレジスター☆

《汎用レジスター》

算術演算・論理演算・比較判断など、各種のデータ操作に使うレジスター。ＡＸ・ＢＸ・ＣＸ・ＤＸの４種類あり、１６ビットの大きさでワード単位の操作をする。バイト単位の操作のために上下、２つに分けて８ビットごとの独立したレジスターとして使うこともできる。

ＡＸ（ＡＨ，ＡＬ）………… アキュムレーター
演算レジスター（アキュムレーター）としての用途が主体。乗除算命令では被除数・被乗数として使われ、結果が入る。入出力命令のデータ受け渡しに使える唯一のレジスターである。イミディエートアドレッシングで使うと、命令が１バイト短くなるなどの特徴をもつ。

ＢＸ（ＢＨ，ＢＬ）………… ベースレジスター
オペランドで指定するメモリーアドレスを修飾するのに使う。ベースアドレスを指示するので、ベースレジスターと呼ぶ。インデックスレジスター（ＳＩ，ＤＩ）と併用すると、２次元配列のテーブル操作が簡単にできる。

ＣＸ（ＣＨ，ＣＬ）………… カウントレジスター
繰り返し処理のループ回数を制御するときに、減数カウンターとして使う。ローテートやシフトなどのビット操作命令では、移動するビット数を指示するのに使う。

ＤＸ（ＤＨ，ＤＬ）………… データレジスター
入出力命令では、入出力ポートのアドレスを指定するのに使う。乗除算命令では、アキュムレーターと併用され、乗算では結果の上位１６ビットが入る。除算では被除数の上位１６ビットとして使われる。

《ポインターレジスターとインデックスレジスター》

ポインターレジスターとインデックスレジスターは、ともにアドレス指定に使う。ポインターレジスターは、スタックセグメントをアクセスするのに使う。インデックスレジスターは単独で、またはベースレジスター（ＢＸ）、ベースポインター（ＢＰ）と組み合わせて、実効アドレスを計算するのに使われる。

ＳＰ………………………… スタックポインター
スタックセグメントのメモリーをプッシュ・ダウン・スタック（後入れ先出し棚）として利用するためのレジスター。スタック命令では自動的に内容が更新され、スタックの先頭番地が常に保持されているようになっている。

ＢＰ………………………… ベースポインター
スタックセグメントのデータをアクセスするとき、ベースアドレスを提供するレジスター。インデックスレジスター（ＳＩまたはＤＩ）と組合せてオペランドアドレスを修飾できる。

ＳＩ ……………………… ソースインデックスレジスター
データをアクセスするときオペランドのアドレスを修飾する。ストリング命令では、移動されるデータのアドレスとして使われる。

ＤＩ ……………………… ディストネーションインデックスレジスター
ＳＩレジスターと同様、オペランドのアドレスを修飾する。ストリング命令では、移動先のアドレスとして使われる。

《セグメントレジスター》

１ＭＢのメモリーのうち、任意の６４ＫＢをセグメントとして扱う。各セグメントの先頭番地をＣＳ、ＤＳ、ＳＳ、ＥＳの４種類のセグメントレジスターで指定する。大きさは１６ビットだが、下位に４ビットの０を加えた２０ビットのセグメントアドレスが提供される。

ＣＳ ……………………… コードセグメントレジスター
プログラム命令の記憶領域を指定するセグメントレジスター。インストラクションポインターと加算され、次に読みだして実行する命令のアドレスが決められる。

ＤＳ ……………………… データセグメントレジスター
プログラムで処理するデータの記憶領域を指定するセグメントレジスター。オペランドで決まる実効アドレスに加算されてデータの物理アドレスが決まる。

ＳＳ ……………………… スタック・セグメントレジスター
プッシュ・ダウン・スタックとして使われるスタック領域を指定するセグメントレジスター。ＳＰやＢＰのポインターレジスターがアドレス指定に使われるときは、スタックセグメントレジスターが自動的に加算され、スタックセグメントのメモリーがアクセスされる。

ＥＳ ……………………… エクストラ・セグメント・レジスター
ストリング操作時、ディストネーションインデックス（ＤＩ）と加算されて、移動先のアドレスが計算される。

《インストラクションポインターとフラグレジスター》

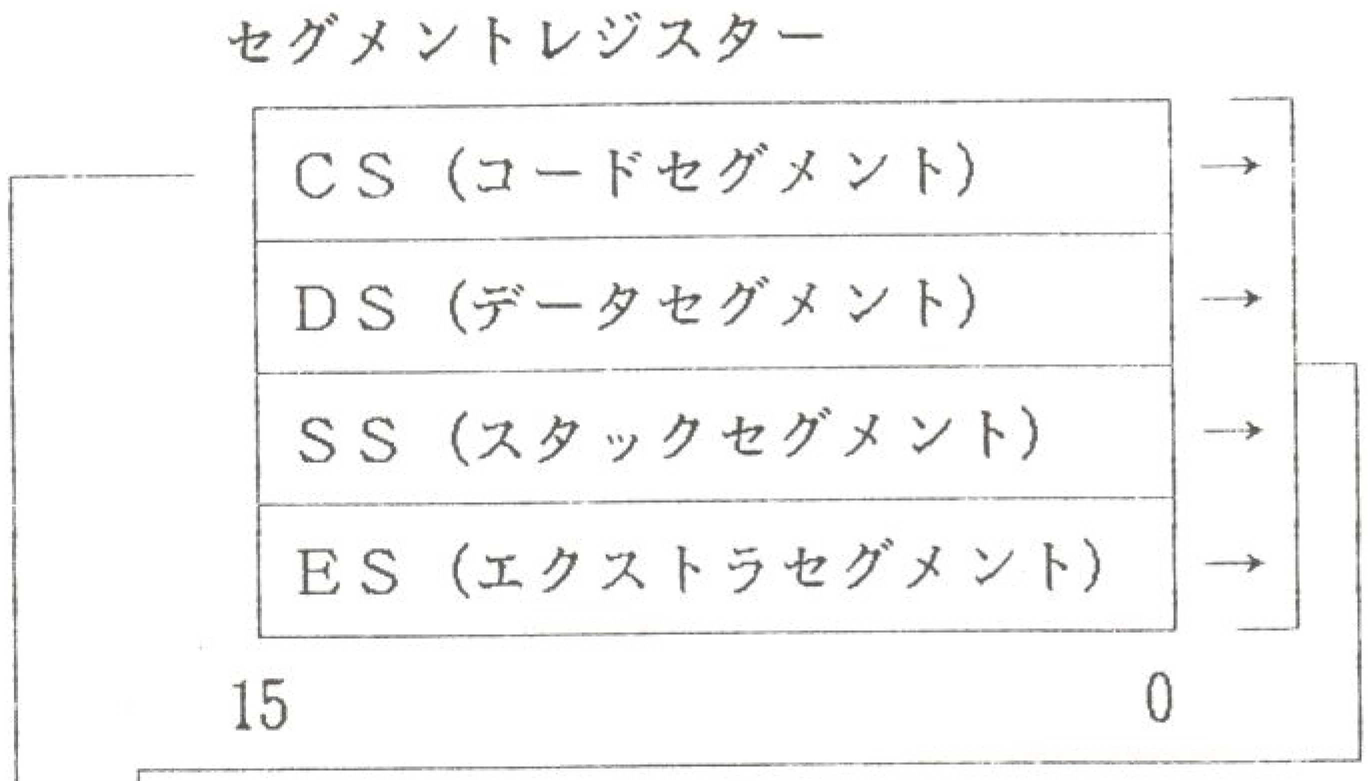
ＩＰ……………………… インストラクションポインター
プログラムカウンターに相当するレジスター。インストラクションを逐次実行するためのアドレスカウンターで、コードセグメントレジスターと組み合わせてインストラクションの物理アドレスを指定する。

Ｆ ……………………… フラグレジスター
演算の結果生じた状態を保持するレジスター。

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
×	×	×	×	O	D	I	T	S	Z	×	A	×	P	×	C

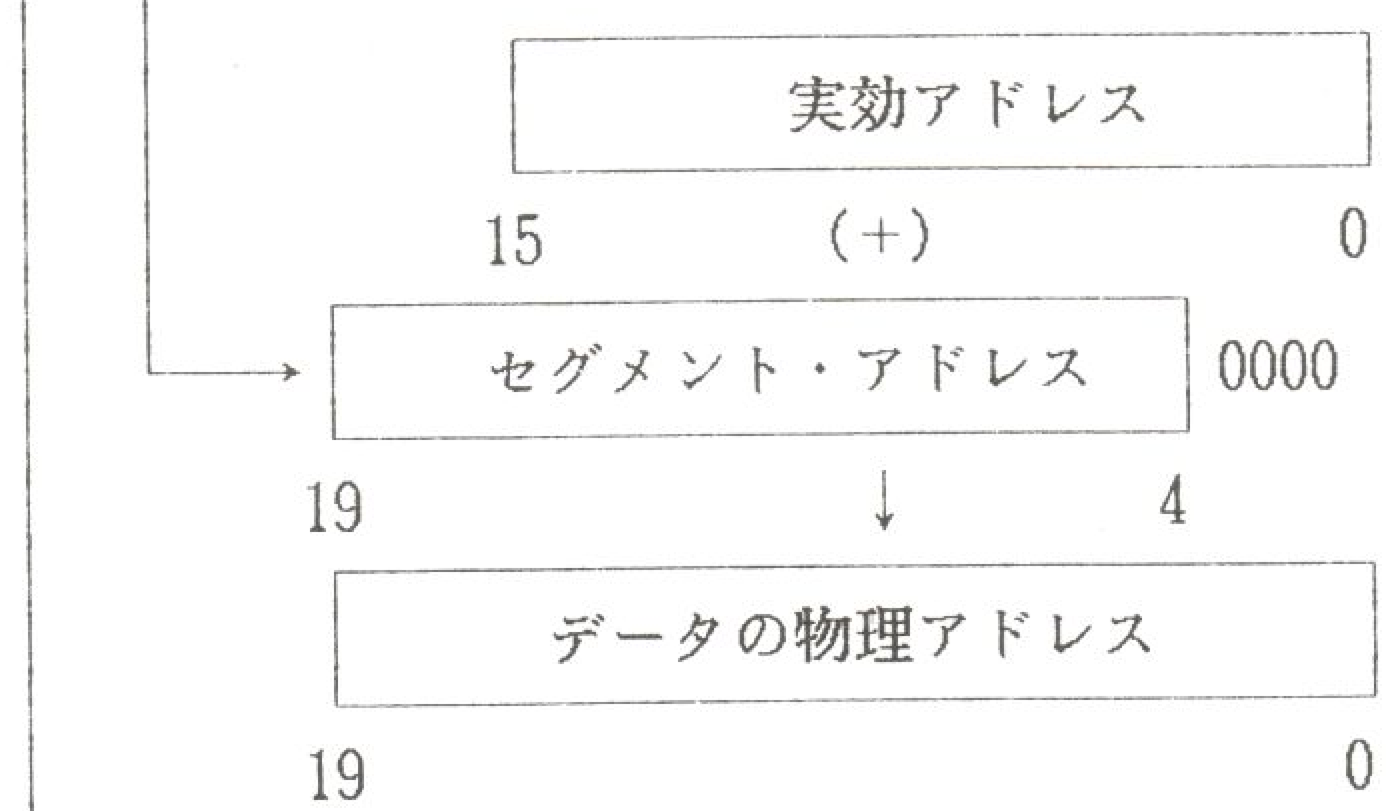
Ｏ：オーバフローフラグ ……… 符号付き２進演算のオーバフロー
Ｄ：ディレクションフラグ ……… インデックスレジスターを自動更新するストリング命令で、０なら増加、１なら減少を指示する
Ｉ：インタラプトフラグ ……… 割り込みを受け付けるには１をセットする
Ｔ：トラップフラグ ……… シングル・ステップの動作モードを設定するフラグ
Ｓ：サインフラグ ……… 結果が負のときセットされる
Ｚ：ゼロフラグ ……… 結果が０のときセットされる
Ａ：補助キャリー ……… ８ビット演算で、下位４ビットからの桁上がりやボロー（桁下がり）が生じたことを示す
Ｐ：パリティーフラグ ……… 結果の下位８ビットに対する奇数パリティー
Ｃ：キャリーフラグ ……… 結果が入る場所の最上位ビットからの桁上がりや減算によるボローが生じたことを示す

☆８０８６　物理アドレスの生成☆



(データの物理アドレス)
命令のオペランドで求められる１６ビットの実効アドレスに、セグメントレジスターを４ビット桁上げした値を加えて、２０ビットの物理アドレスを計算する。どのセグメントレジスターを使うかは、オペランドで決まる。ただし、セグメントプレフィックスで指定すれば変更できる。

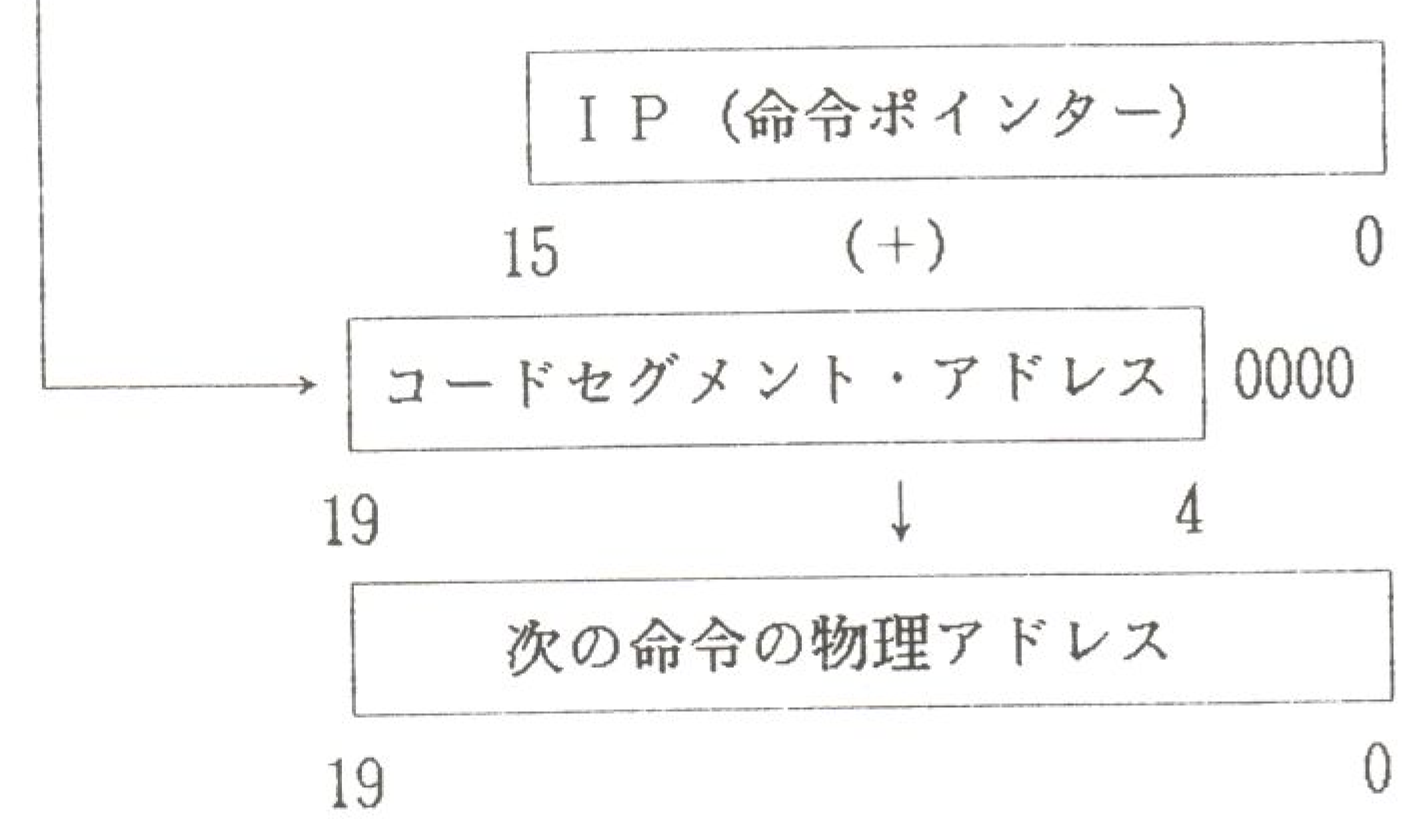
オペランドの実効アドレス	セグメントレジスター
Ｄｉｓｐ	ＤＳ
ＢＸ	ＤＳ
ＳＩ	ＤＳ
ＤＩ	ＤＳ
ＢＸ＋Ｄｉｓｐ	ＤＳ
ＳＩ＋Ｄｉｓｐ	ＤＳ
ＤＩ＋Ｄｉｓｐ	ＤＳ
ＢＰ＋Ｄｉｓｐ	ＳＳ
ＢＸ＋ＳＩ	ＤＳ
ＢＸ＋ＤＩ	ＤＳ
ＢＰ＋ＳＩ	ＳＳ
ＢＰ＋ＤＩ	ＳＳ
ＢＸ＋ＳＩ＋Ｄｉｓｐ	ＤＳ
ＢＸ＋ＤＩ＋Ｄｉｓｐ	ＤＳ
ＢＰ＋ＳＩ＋Ｄｉｓｐ	ＳＳ
ＢＰ＋ＤＩ＋Ｄｉｓｐ	ＳＳ



(次の命令の物理アドレス)
プログラムの命令の物理アドレスは、命令ポインターに、コードセグメントを４ビット桁上げした値を加えて、２０ビットの物理アドレスを計算する。

次に実行する命令のある実効アドレス

分岐命令以外の場合
　ＩＰ　＋　命令のバイト数
分岐命令の場合
　Ｄｉｓｐ
　ＩＰ　＋　Ｄｉｓｐ



注意　変位（Ｄｉｓｐ）は、１バイトと２バイトの２種類ある。
　１バイトの場合、最上位ビットを符号として２バイトに拡張してから計算する。
　例　８５ＨならＦＦ８５Ｈの意味になる

8. 9801 機械語の説明

ここでは機械語を機能別に分類し、ニーモニックの表記方法を一覧表にまとめた。機械語を初めて学ぶ読者は、まず「ＵＳＲ関数とＣＡＬＬ文の使い方」でＢＡＳＩＣと機械語の関係をつかみ、「機械語プログラム集」を試してからこの章を読むとよい。

☆表の読み方☆

ニーモニック	フラグ ＯＳＺＡＰＣ	説　　明
《転送命令》		
ＭＯＶ　adr ,acum	－－－－－－	move　データの転送

move → ニーモニックのもとになった英語と、命令の説明

－－－－－－ → 命令の実行によるフラグの変化
Ｓ：セット　（1になる）
Ｒ：リセット（0になる）
Ｘ：不定
Ｏ：演算結果による
－：不変
（Ｉ・Ｄ・Ｔのフラグは通常の命令では変化しない。変化するときだけ説明してある）

acum → ソースオペランド

adr → デスティネーションオペランド
操作の対象を表す

ＭＯＶ → オペコード
操作を表す

☆オペランドの表現☆

reg　汎用レジスター、ポインターレジスター、インデックスレジスター。汎用レジスターはバイトまたはワードの指定ができる
regw　１ワードの汎用レジスター、ポインターレジスター、インデックスレジスター
seg　セグメントレジスター
acum　アキュムレーター。ＡＸ、ＡＬレジスター
data　データ。4桁または2桁の16進数。〔　〕で囲まない
jadr　とびさきの番地。4桁の16進数。〔　〕で囲まない（１５３ページ参照）
adr　メモリーの番地。直接指定、間接指定がある
オペランドの書き方により物理アドレスの計算に使うセグメントレジスターが決まる

（直接指定）

〔ｎｎｎｎ〕	ＤＳ＊10H ＋ｎｎｎｎ番地

ｎｎｎｎは4桁の16進数　〔　〕で囲む

（間接指定）

ｎｎｎｎ〔ＢＸ＋ＳＩ〕	ＤＳ＊10H ＋ＢＸ＋ＳＩ＋ｎｎｎｎ番地
ｎｎｎｎ〔ＢＸ＋ＤＩ〕	ＤＳ＊10H ＋ＢＸ＋ＤＩ＋ｎｎｎｎ番地
ｎｎｎｎ〔ＢＰ＋ＳＩ〕	ＳＳ＊10H ＋ＢＰ＋ＳＩ＋ｎｎｎｎ番地
ｎｎｎｎ〔ＢＰ＋ＤＩ〕	ＳＳ＊10H ＋ＢＰ＋ＤＩ＋ｎｎｎｎ番地
ｎｎｎｎ〔ＳＩ〕	ＤＳ＊10H ＋ＳＩ＋ｎｎｎｎ番地
ｎｎｎｎ〔ＤＩ〕	ＤＳ＊10H ＋ＤＩ＋ｎｎｎｎ番地
ｎｎｎｎ〔ＢＸ〕	ＤＳ＊10H ＋ＢＸ＋ｎｎｎｎ番地
ｎｎｎｎ〔ＢＰ〕	ＳＳ＊10H ＋ＢＰ＋ｎｎｎｎ番地

ｎｎｎｎ（変位）は4桁または2桁の16進数。変位がゼロのときは指定しない

・セグメントレジスターの変更
　セグメントレジスターにコロンを付けたセグメントプレフィックスを命令または adrの直前に書く

（例）
ＥＳ：	
ＭＯＶ　〔100〕，ＡＸ	ＥＳ＊10H ＋100H番地
ＭＯＶ　ＳＳ：〔１００〕，ＡＸ	ＳＳ＊10H ＋100H番地

・バイト、ワードの区別
　指定するオペランドが1バイトなのか1ワード（2バイト）なのか区別できない場合には、オペランドの前にＢＹＴＥまたはＷＯＲＤをつけて区別する。

（例）
ＭＯＶ　ＢＹＴＥ　〔１００〕，１０	100H番地に10をいれる
ＭＯＶ　ＷＯＲＤ　〔１００〕，１０	100H,101H 番地に10をいれる

ニーモニック		フラグ OSZAPC	説　　明
《転送命令》			
MOV	adr ,acum acum,adr reg1,reg2 reg ,adr adr ,reg seg ,reg seg ,adr reg ,seg adr ,seg reg ,data adr ,data	−−−−−−	move データの転送 ソースオペランドの内容をデスティネーションオペランドに入れる。デスティネーションオペランドにCS（コードセグメント）レジスターは指定できない MOV AX,〔1000〕 1000H 番地の内容をAXレジスターに入れる MOV AX,1000 AXレジスターに1000H を入れる
XCHG	acum,reg reg1,reg2 adr ,reg	−−−−−−	exchange データの入れ換え ソースオペランドとデスティネーションオペランドの内容を入れ換える
XLAT		−−−−−−	translate メモリーの（BX+AL）番地の内容をALに入れる。AL←（BX+AL）
LAHF		−−−−−−	load AH from flags SF・ZF・AF・PF・CF（フラグレジスターの下位バイト）をAHに入れる。 AH←SZxAxPxC
SAHF		−OOOOO	store AH to flag AHをフラグにセット LAHFの逆。SZxAxPxC←AH
LEA	regw,adr	−−−−−−	load effective adress アドレスの値をレジスターに移す。reg←（adrの値）
LDS	regw,adr	−−−−−−	load pointer using DS メモリーの指定した番地から2バイトをレジスターに、次の2バイトをDSレジスターに移す reg←adr、DS←（adr+2）
LES	regw,adr	−−−−−−	load pointer using ES メモリーの指定した番地から2バイトをレジスターに、次の2バイトをESレジスターに移す reg←adr、ES←（adr+2）
PUSH	regw seg adr	−−−−−−	push word on stack レジスターやメモリーをスタックに退避。SPを−2して、SPの示すメモリーにレジスターやメモリーの内容を移す SP←SP−2、（SP）←regまたはadr
POP	regw seg adr	−−−−−−	pop word off stack スタックからの取り出し。SPの示すメモリーの内容をレジスターやメモリーに移し、SPに2を加える。CSレジスターは使えない。PUSHの逆 regまたはadr←（SP）、SP←SP+2
PUSHF		−−−−−−	push flag onto stack フラグをスタックに退避 SPを−2して、SPの示すメモリーにフラグを移す。SP←SP−2、（SP）←フラグ

ニーモニック		フラグ OSZAPC	説　　明
POPF		OOOOOO	pop flag off stack　フラグをスタックから取り出す。SPの示すメモリーの内容をフラグに移しSPに2を加える。PUSHFの逆 FLAG←（SP）、SP←SP+2
《入出力命令》			
IN	acum,port acum,DX	-------	input　ポート番号またはDXレジスターが示すポートから、アキュムレーターにデータを入力
OUT	port,acum DX,acum	-------	output　ポート番号またはDXレジスターが示すポートへ、アキュムレーターのデータを出力
《算術演算命令》			
ADD	reg1，reg2 reg　，adr adr　，reg acum，data reg　，data adr　，data	OOOOOO	addition　加算 ソースオペランドの内容とデスティネーションオペランドの内容を加算し、デスティネーションオペランドに入れる ADD　BX，SI SIレジスターの内容をBXレジスターに加算
ADC	reg1，reg2 reg　，adr adr　，reg acum，data reg　，data adr　，data	OOOOOO	add with carry　加算命令ADDにCF（キャリーフラグ）の加算を加えたもの。1バイト（ワード）の加算（ADD）の結果、桁あふれが生じるとCFが1になる。2バイト（ワード）以上の加算で桁上げするときに、この命令を使う ADC　BX，〔SI〕 SIレジスターで指定したメモリーの内容とCFをBXレジスターに加算
INC	reg adr	OOOOO-	increment　インクリメント reg←reg+1 adr←adr+1
AAA		XXXOXO	ascii adjust for add　アンパック10進数どうしを直接加算する命令はないので、ADD命令で加算する。アキュムレーターにある計算結果を、この命令でアンパック10進数に戻す
DAA		XOOOOO	decimal adjust for add　パック10進数どうしを直接加算する命令はないので、ADD命令で加算する。アキュムレーターにある計算結果を、この命令でパック10進数に戻す
SUB	reg1，reg2 reg　，adr adr　，reg acum，data reg　，data adr　，data	OOOOOO	subtract　減算 デスティネーションオペランドの内容からソースオペランドの内容を減算し、デスティネーションオペランドに入れる SUB　BL，10 BLレジスターから10Hを減算

ニーモニック		フラグ OSZAPC	説　　明
SBB	reg1，reg2 reg　，adr adr　，reg acum，data reg　，data adr　，data	OOOOOO	subtract with borrow　減算命令ＳＵＢにＣＦ（キャリーフラグ）の減算を加えたもの ＡＤＣの減算版 ＳＢＢ　ＢＨ，１０ ＢＨレジスターから10H とＣＦを減算
DEC	reg adr	OOOOO－	decrement　デクリメント ｒｅｇ←ｒｅｇ－１ ａｄｒ←ａｄｒ－１
NEG	reg adr	OOOOOS	negate　正負の反転 レジスターやメモリーの内容を２の補数にする
CMP	reg1，reg2 reg　，adr adr　，reg acum，data reg　，data adr　，data	OOOOOO	compare　比較命令 デスティネーションオペランドの内容からソースオペランドの内容を減算した結果により、フラグを変化させる。デスティネーションオペランドの内容は変化しない ＣＭＰ　〔ＢＸ〕，ＣＸ ＢＸレジスターで指定したメモリーの内容とＣＸレジスターの比較
AAS		XXXOXO	ascii adjust for subtract　アンパック１０進数どうしを直接減算する命令はないので、ＳＵＢ命令で減算する。アキュムレーターにある計算結果を、この命令でアンパック１０進数に戻す
DAS		XOOOOO	decimal adjust for subtract　パック１０進数どうしを直接減算する命令はないので、ＳＵＢ命令で減算する。アキュムレーターにある計算結果を、この命令でパック１０進数に戻す
MUL	reg adr	OXXXXO （桁上げの時Ｃ・Ｏ←１）	multiply　乗算 ワード計算ではＡＸレジスターとの乗算になり結果はＤＸとＡＸの４バイト、バイト計算はＡＬとの乗算になり結果はＡＨとＡＬの２バイトに入る （ワード計算） ＤＸとＡＸ←ＡＸ＊（ｒｅｇ　または　ａｄｒ） （バイト計算） ＡＸ←ＡＬ＊（ｒｅｇ　または　ａｄｒ）
IMUL	reg adr	OXXXXO	integer multiply　符号付き乗算 ワード計算では 8000H～FFFFH 、バイト計算では 80H～FFH をマイナスの値として扱う。計算動作はＭＵＬと同じ
AAM		XOOXOX	ascii adjust for multiply　アンパック１０進数どうしを直接乗算する命令はないので、ＭＵＬ命令で乗算する。アキュムレーターにある計算結果を、この命令でアンパック１０進数に戻す

ニーモニック		フラグ OSZAPC	説　　明
DIV	reg adr	XXXXXX	divide　除算 （ワードの計算） AX←DXとAX／（reg又はadr）の商 DX←DXとAX／（reg又はadr）の剰余 （バイトの計算） AL←AX／（regまたはadr）の商 AH←AX／（regまたはadr）の剰余 ゼロで割ると商が入るレジスターがオーバーフローして、タイプ0の割り込み（除算エラー）が起こる
IDIV	reg adr	XXXXXX	integer divide　符号付き除算 ワード計算では 8000H～FFFFH 、バイト計算では 80H～FFH をマイナスの値として扱う。計算動作はDIVと同じ
AAD		XOOXOX	ascii adjust for divide　アンパック１０進数を直接除算する命令はないので、DIV命令で除算する。あらかじめアキュムレーターにある被除数を、この命令で変換しておく。DIV命令のバイト計算で商、余りはアンパック１０進数になる
CBW		-------	convert byte to word ALをAXに符号を付けて拡張する
CWD		-------	convert word to double-word AXをDXとAXに符号を付けて拡張する
《論理演算命令》			
NOT	reg adr	-------	not　NOT演算。１の補数にする reg又はadr←NOT（reg又はadr）
AND	reg1，reg2 reg　，adr adr　，reg acum，data reg　，data adr　，data	ROOXOR	and　AND演算 ソースオペランドとデスティネーションオペランドの内容をAND演算し、結果をデスティネーションオペランドに入れる AND　DX，〔SI〕 SIレジスターで指定したメモリーとDXレジスターのAND演算
OR	reg1，reg2 reg　，adr adr　，reg acum，data reg　，data adr　，data	ROOXOR	or　OR演算 ソースオペランドとデスティネーションオペランドの内容をOR演算し、結果をデスティネーションオペランドに入れる OR　DX，１０〔SI〕 （SIレジスターの値＋10H ）番地のメモリーとDXレジスターのOR演算
XOR	reg1，reg2 reg　，adr adr　，reg acum，data reg　，data adr　，data	ROOXOR	exclusive or　XOR演算 ソースオペランドとデスティネーションオペランドの内容をXOR演算し、結果をデスティネーションオペランドに入れる XOR　DX，１０００〔SI〕 （SIレジスターの値＋1000H ）番地のメモリーとDXレジスターのXOR演算

ニーモニック	フラグ OSZAPC	説　　明
TEST　reg1，reg2 reg　，adr adr　，reg acum，data reg　，data adr　，data	ROOXOR	test　AND演算の結果でフラグだけを変化させる。デスティネーションオペランドの内容は変化しない TEST　10〔SI〕，DX DXレジスターと（SIレジスターの値+10H）番地のメモリーとのTEST
SHL　または　SAL reg　，1 または CL adr　，1 または CL	OOOXOO	shift logical left／shift arithmetic left 左シフト命令。1またはCLの値だけレジスターやメモリーのビットを左に移動。最上位ビットはCFへ、最下位ビットは0が埋まる
SHR　reg　，1 または CL adr　，1 または CL	OOOXOO	shift logical right　右シフト命令 1またはCLの値だけレジスターやメモリーのビットを右に移動。最下位ビットはCFへ、最上位ビットは0が埋まる
SAR　reg　，1 または CL adr　，1 または CL	OOOXOO	shift arithmetic right　右シフト命令 SHR命令とのちがいは最上位ビットに0が埋まらず、もとのまま変わらない
ROL　reg　，1 または CL adr　，1 または CL	O－－－－O	rotate left　左回転命令 1またはCLの値だけレジスターやメモリーのビットを左に回転。最上位ビットはCFに入るだけでなく最下位ビットにも回る
ROR　reg　，1 または CL adr　，1 または CL	O－－－－O	rotate right　右回転命令 最下位ビットはCFに入るだけでなく最上位ビットにも回る
RCL　reg　，1 または CL adr　，1 または CL	O－－－－O	rotate left through carry CFを通してレジスターやメモリーの左回転。最上位ビットはCFに、CFは最下位ビットに回る
RCR　reg　，1 または CL adr　，1 または CL	O－－－－O	rotate right through carry CFを通してレジスターやメモリーの右回転。最下位ビットはCFに、CFは最上位ビットに回る
《ストリング操作命令》		
		CXレジスターを1減らしながら、次に続くストリング操作命令を条件が成立するまで繰り返す。次頁のストリング操作命令と組み合わせて使う
REP　　または REPZ　　または REPE	－－－－－－	repeat repeat while zero repeat while equal 条件　CX＝0　または　ZF＝0
REPNZ　または REPNE	－－－－－－	repeat while not zero repeat while not equal 条件　CX＝0　または　ZF＝1

ニーモニック	フラグ OSZAPC	説　　明
MOVSB MOVSW	−−−−−−−	move string - byte／word　転送 データセグメント内のＳＩ（ソースインデックス）が示すメモリーの内容をエクストラセグメント内のＤＩ（ディストネーションインデックス）が示すメモリーに移す。ＤＦが0であればＳＩ、ＤＩをインクリメント、ＤＦが1であればデクリメントする。ＭＯＶＳＢはバイト、ＭＯＶＳＷはワードの転送（以下同じ） （ＥＳ＊10H ＋ＤＩ）←（ＤＳ＊10H ＋ＳＩ） （ＳＩ）←（ＳＩ）±ｎ（Ｂなら1、Ｗなら2） （ＤＩ）←（ＤＩ）±ｎ（Ｂなら1、Ｗなら2）
CMPSB CMPSW	OOOOOO	compare string-byte ／word　比較 データセグメント内のＳＩが示すメモリーの内容からエクストラセグメント内のＤＩが示すメモリーの内容を減算しフラグのみ変化させる （ＤＳ＊10H ＋Ｓ！）−（ＥＳ＊10H ＋ＤＩ） （ＳＩ）←（ＳＩ）±ｎ（Ｂなら1、Ｗなら2） （ＤＩ）←（ＤＩ）±ｎ（Ｂなら1、Ｗなら2）
SCASB SCASW	OOOOOO	scan string - byte／word　比較 アキュムレーターからエクストラセグメント内のＤＩが示すメモリーの内容を減算しフラグのみ変化させる （ＡＸ　または　ＡＬ）−（ＥＳ＊10H ＋ＤＩ） （ＤＩ）←（ＤＩ）±ｎ（Ｂなら1、Ｗなら2）
LODSB LODSW	−−−−−−−	load string - byte／word　転送 データセグメント内のＳＩが示すメモリーの内容をアキュムレーターに移す。ＲＥＰによる繰り返しは無意味 （ＡＸ　または　ＡＬ）←（ＤＳ＊10H ＋ＳＩ） （ＳＩ）←（ＳＩ）±ｎ（Ｂなら1、Ｗなら2）
STOSB STOSW	−−−−−−−	store string - byte ／word　転送 アキュムレーターの内容をエクストラセグメント内のＤＩが示すメモリーに移す （ＥＳ＊10H ＋ＤＩ）←（ＡＸ　または　ＡＬ） （ＤＩ）←（ＤＩ）±ｎ（Ｂなら1、Ｗなら2）
《分岐命令》		
JMP 　jadr 　reg 　adr	−−−−−−−	jump　ジャンプ 指定した番地にとぶ レジスターの値が示す番地にとぶ メモリーの内容が示す番地にとぶ モニターのアセンブラーではセグメントをまたがるＪＭＰは書けない。機械語（OP-CODE はＥＡかＦＦ、jadrは 153ページ参照）で入れる
JMPS　jadr	−−−−−−−	jump short　指定できるとびさき番地は現在の番地−128 から+127 の範囲
JCXZ　jadr	−−−−−−−	jump if CX zero ＣＸレジスターの値が0の時、指定した番地へとぶ。指定できる番地は現在の番地−128 から+127 の範囲

ニーモニック	フラグ ＯＳＺＡＰＣ	説　　明
   ＪＥ　　または　ＪＺ jadr ＪＮＥ　または　ＪＮＺ jadr ＪＬ　　または　ＪＮＧＥ jadr ＪＮＬ　または　ＪＧＥ jadr ＪＬＥ　または　ＪＮＧ jadr ＪＮＬＥ　または　ＪＧ jadr ＪＢ　　または　ＪＮＡＥ jadr ＪＮＢ　または　ＪＡＥ jadr ＪＢＥ　または　ＪＮＡ jadr ＪＮＢＥ　または　ＪＡ jadr ＪＰ　　または　ＪＰＥ jadr ＪＮＰ　または　ＪＰＯ jadr ＪＯ　　　jadr ＪＮＯ　　jadr ＪＳ　　　jadr ＪＮＳ　　jadr	－－－－－－－	条件ジャンプ。フラグがそれぞれの条件を満足した時、指定した番地へとぶ。指定できる番地は現在の番地－128 から＋127 の範囲 jump equal／jump zero 条件　ＺＦ＝１ jump not equal／jump not zero 条件　ＺＦ＝０ jump less than／jump not greater equal 条件　ＳＦ　ＸＯＲ　ＯＦ＝１ jump not less than／jump gteater equal 条件　ＳＦ　ＸＯＲ　ＯＦ＝０ jump less equal ／jump not greater than 条件（ＳＦ　ＸＯＲ　ＯＦ）ＯＲ　ＺＦ＝１ jump not less equal ／jump greater than 条件（ＳＦ　ＸＯＲ　ＯＦ）ＯＲ　ＺＦ＝０ jump bellow ／jump not above equal 条件　ＣＦ＝１ jump not bellow ／jump above equal 条件　ＣＦ＝０ jump bellow equal ／jump not above 条件　ＣＦ　ＯＲ　ＺＦ＝１ jump not bellow equal ／jump above 条件　ＣＦ　ＯＲ　ＺＦ＝０ jump parity ／jump parity even 条件　ＰＦ＝１ jump not parity ／jump parity odd 条件　ＰＦ＝０ jump overflow　　　条件　ＯＦ＝１ jump not overflow　　条件　ＯＦ＝０ jump sign　　　　　条件　ＳＦ＝１ jump not sign　　　　条件　ＳＦ＝０
    ＬＯＯＰ　　jadr ＬＯＯＰＮＺまたは　ＬＯＯＰＮＥ jadr ＬＯＯＰＺ　または　ＬＯＯＰＥ jadr	－－－－－－－	繰り返し命令 ＣＸを１減らしてから条件が成立していれば、指定した番地へとぶ。指定できる番地は現在の番地－128 から＋127 の範囲 loop　　　条件　ＣＸ≠０ loop not zero ／loop not equal 条件　ＣＸ≠０　かつ　ＺＦ＝０ loop zero ／loop equal 条件　ＣＸ≠０　かつ　ＺＦ＝１
ＣＡＬＬ jadr reg adr	－－－－－－－	call　サブルーチンの呼び出し jadrまたはreg またはadr で指定したメモリーの番地からはじまるサブルーチンを呼び出す。復帰する番地（この命令の次の番地）はスタックに入る。セグメントをまたがるＣＡＬＬは機械語（OP-CODE は９ＡかＦＦ、jadrは 153ページ参照）で入れる
ＲＥＴ　　(data)	－－－－－－－	return　サブルーチンから復帰。dataを記入するとＳＰにその値を加算。セグメンをまたがるＲＥＴは機械語（OP-CODE はＣＢ）で入れる
ＩＮＴ　　type (注)	－－－－－－－	interrupt　typeで指定した割り込みベクトル番号に対応する割り込み処理にとぶ。フラグはスタックに退避する。ＩＦとＴＦは０になる

ニーモニック	フラグ OSZAPC	説　　明
INTO	－－－－－－	interrupt overflow　OFが1のときtype 4H の割り込み（オーバーフロー）が起こる
IRET	OOOOOO	interrupt return　割り込みからの復帰 退避したフラグはスタックから戻る
《CPU制御命令》		
		フラグのセット、クリアー
STD	－－－－－－	set direction flag　DFのセット DF←1
CLD	－－－－－－	clear direction flag　DFのクリアー DF←0
STI	－－－－－－	set interrupt flag　IFのセット IF←1
CLI	－－－－－－	clear interrupt flag　IFのクリアー IF←0
STC	－－－－－S	set carry flag　CFのセット CF←1
CLC	－－－－－R	clear carry flag　CFのクリアー CF←0
CMC	－－－－－O	complement carry flag　CFの反転 CF←NOT（CF）
NOP	－－－－－－	no operation 何もしないで次の命令に行く
HLT	－－－－－－	halt　外部割り込みが発生、またはリセットされるまでcpuを待ち状態にする
WAIT	－－－－－－	wait　外部のハードウエアと同期をとるためcpuを待ち状態にする
ESC　命令，reg ／adr	－－－－－－	escape 数値データプロセッサーの制御に使う
LOCK	－－－－－－	lock bus　バスロック信号を出力して、この命令の次の命令を実行している間は、バスを他のプロセッサーから使用できないようにする
《セグメントプレフィックス》		
ES： CS： SS： DS：	－－－－－－	オペランドの物理アドレス計算に使うセグメントレジスターを変更する。命令またはオペランドの直前で指定する

（注）割り込みベクトルテーブルは、割り込み処理ルーチンを呼び出すためのテーブルで、00000H番地から003FFH番地に4バイトずつとび先の番地が登録してある。下位2バイトでコードセグメントベース、上位2バイトでオフセットを表す。テーブルの4バイトごとに 00Hから FFHまで256の番号を付け、これをベクトル番号と呼ぶ。

INT命令のtypeでベクトル番号を指定すれば、対応する割り込み処理ルーチンを呼び出す。

システムが使う割り込み処理ルーチンを不用意に使うと誤動作するおそれがあるので、通常ユーザーが使えるのは次のものに限る。ただし、割り込み処理ルーチンはユーザーが準備しておく必要がある。

・ベクトルテーブルの番地	ベクトル番号
00100H～001FFH	40H～7FH
003C4H～003FFH	F1H～FFH

9. 9801　ニーモニックと機械語の対応表

《機械語コードの生成》

機械語は同じ命令でも、レジスターやメモリーの指定により、命令の長さが変わる。アセンブラーニーモニックのオペレーションコードとオペランドの形式により４種類に分類できる。オペレーションコードにより，１バイト目が決まり、オペランドで２バイト目以降を生成する。
２バイト目にレジスターやメモリー番地の指定をする場合、ｍｏｄ、ｒｅｇ、ａｄｒの３つの２進数パラメーターを組み合わせて作る。

《レジスターとメモリーの２進数パラメーター》

レジスター

reg	mod ＝１１ byte	mod ＝１１ word
000	ＡＬ	ＡＸ
001	ＣＬ	ＣＸ
010	ＤＬ	ＤＸ
011	ＢＬ	ＢＸ
100	ＡＨ	ＳＰ
101	ＣＨ	ＢＰ
110	ＤＨ	ＳＩ
111	ＢＨ	ＤＩ

00	ＥＳ
01	ＣＳ
10	ＳＳ
11	ＤＳ

メモリー

adr	mod ＝ ００	０１	１０
000	〔BX＋SI〕	nn 〔BX＋SI〕	nnnn 〔BX＋SI〕
001	〔BX＋DI〕	nn 〔BX＋DI〕	nnnn 〔BX＋DI〕
010	〔BP＋SI〕	nn 〔BP＋SI〕	nnnn 〔BP＋SI〕
011	〔BP＋DI〕	nn 〔BP＋DI〕	nnnn 〔BP＋DI〕
100	〔SI〕	nn 〔SI〕	nnnn 〔SI〕
101	〔DI〕	nn 〔DI〕	nnnn 〔DI〕
110	〔nnnn〕	nn 〔BP〕	nnnn 〔BP〕
111	〔BX〕	nn 〔BX〕	nnnn 〔BX〕

nn　：１バイトの変位（Ｄｉｓｐ）
nnnn　：２バイトの変位（Ｄｉｓｐ）
〔nnnn〕：アドレスの直接指定

《機械語の形式》

（1）形式１

レジスターとレジスター、レジスターとメモリーの転送や演算などオペランドを２つ指定する形式
オペランド：　reg ，reg ／adr （reg ／adr はレジスターまたはメモリーを意味する）
　　　　　　　reg ／adr ，reg

１バイト目・・・・・オペレーションコード（ＯＰ－ＣＯＤＥ）
２バイト目・・・・・レジスター、メモリーの指定

b7 b6	b5 b4 b3	b2 b1 b0
ｍｏｄ	ｒｅｇ	reg ／adr

- reg ／adr：オペランドのreg またはadr を指定
- reg：オペランドのreg を指定
  ８Ｃ・８ＥのＭＯＶ命令はseg を指定
  (b5に0 、b4とb3にセグメントレジスターを指定)
- mod：reg またはadr のモード（ mod ）を指定

３バイト目以降・・・メモリー番地の指定に変位がある時、１または２バイトの変位を付ける
　　　　　　　　　　変位は、下位バイトと上位バイトが逆になる

・reg とreg （mod ＝11）

OP-CODE	11reg reg

(例) ＡＤＤ　ＢＸ，ＳＩ
reg ＝BX＝011,reg ＝SI＝110
２バイト目＝（11011110）＝ＤＥ
機械語：０３ＤＥ

・reg とadr 直接指定（mod ＝00，adr ＝110 ）

OP-CODE	00reg 110	disp-L	disp-H

(disp-Lは変位の下位バイト、disp-Hは上位バイト)

(例) ＡＤＤ　ＢＸ，〔１０００〕
reg ＝BX＝011
２バイト目＝（00011110）＝１Ｅ
機械語：０３１Ｅ００１０

・reg とadr 間接指定－変位なし（mod ＝00，adr ≠110 ）

OP-CODE	00reg adr

（例）ＡＤＤ　ＢＸ，〔ＳＩ〕
reg ＝BX＝011,adr ＝〔SI〕＝100
２バイト目＝（00011100）＝１Ｃ
機械語：０３１Ｃ

・reg とadr 間接指定－変位 1byte（mod ＝01）

OP-CODE	01reg adr	disp

（例）ＡＤＤ　ＢＸ，１０〔ＳＩ〕
reg ＝BX＝011,adr ＝〔SI〕＝100
２バイト目＝（01011100）＝５Ｃ
機械語：０３５Ｃ１０

・reg とadr 間接指定－変位 2byte（mod ＝10）

OP-CODE	10reg adr	disp-L	disp-H

（disp-Lは変位の下位バイト、disp-Hは上位バイト）

（例）ＡＤＤ　ＢＸ，１０００〔ＳＩ〕
reg ＝BX＝011,adr ＝〔SI〕＝100
２バイト目＝（10011100）＝９Ｃ
機械語：０３９Ｃ００１０

（２）形式２

レジスターまたはメモリーへデータの転送、データとの演算など。または１バイト目の命令で使用するレジスターが決まるもの。

オペランド：　reg ／adr ，data
　　　　　　　reg ／adr
　　　　　　　REG ，reg ／adr　　　（REG は機械語コード表のオペランド指定で、
　　　　　　　reg ／adr ，REG　　　 大文字で表したレジスター）

１バイト目・・・・・オペレーションコード　　２バイト目のb5～b3の３ビットとあわせて１つの命令になる

２バイト目・・・・・レジスター、メモリーの指定

b7 b6	b5 b4 b3	b2 b1 b0
ｍｏｄ	ｘｘｘ	reg ／adr

reg ／adr：オペランドのreg またはadr を指定
ｘｘｘ：命令の一部
ｍｏｄ：reg またはadr のモード指定

３バイト目以降・・・メモリー番地の指定に変位がある時、１または２バイトの変位を付ける
データがあれば、次にバイトの処理では１バイト、ワードの処理では２バイトを続ける。データは下位バイトと上位バイトが逆になる

・reg　（mod ＝11）

OP-CODE	11xxx reg	data-L	data-H

（data-Lはデータの下位バイト、data-Hは上位バイト）

（例）ＡＤＤ　ＢＸ，１０００
xxx ＝ADD ＝000,reg ＝BX＝011
２バイト目＝（11000011）＝Ｃ３
機械語：８１Ｃ３００１０

・adr 直接指定（mod ＝00，adr ＝110 ）

OP-CODE	00xxx 110	disp-L	disp-H	data-L	data-H

（例）ＡＤＤ　ＷＯＲＤ　〔２０００〕，１０００
xxx ＝ADD ＝000
２バイト目＝（00000110）＝０６
機械語：８１０６００２０００１０

・adr 間接指定－変位なし（mod ＝00，adr ≠110 ）

OP-CODE	00xxx adr	data-L	data-H

（例）ＡＤＤ　ＷＯＲＤ〔ＢＸ〕，１０００
xxx ＝ADD ＝000,adr ＝〔BX〕＝111
２バイト目＝（00000111）＝０７
機械語：８１０７００１０

・adr 間接指定－変位 1byte (mod ＝01)

OP-CODE	01xxx adr	disp	data-L	data-H

(例) ＡＤＤ　ＷＯＲＤ　２０〔ＢＸ〕，１０００
xxx ＝ADD ＝000,adr ＝〔BX〕＝111
２バイト目＝ (01000111) ＝４７
機械語：８１４７２０００１０

・adr 間接指定－変位 2byte (mod ＝10)

OP-CODE	10xxx adr	disp-L	disp-H	data-L	data-H

(例) ＡＤＤ　ＷＯＲＤ　２０００〔ＢＸ〕，１０００
xxx ＝ADD ＝000,adr ＝〔BX〕＝111
２バイト目＝ (10000111) ＝８７
機械語：８１８７００２０００１０

（３）形式３

レジスターやメモリーを指定せず、データ、変位、ポート番号、割り込みベクトル番号を指定。
または、オペレーションコードの１バイト目で使用するレジスターが決まるもの。
オペランド：　data　または　jadr　または　port　または　type
REG ,data　または　jadr　または　port　または　type
(REG は機械語コード表のオペランド指定で、大文字で表したレジスター)

１バイト目・・・・・オペレーションコード　　(レジスターの指定を含む場合もある)
２バイト目以降・・・データ、変位、メモリーのとびさき番地を１または２バイトで指定

・data

OP-CODE	data-L	data-H

(例) ＡＤＤ　ＡＸ，１０００
機械語：０５００１０

・jadr

OP-CODE	disp-L	disp-H	seg-L	seg-H

(seg-L はセグメントベースの下位バイト、seg-H は上位バイト)

(例) 1000　ＪＥ　１０５２
IP＝1002，jadr＝1052
IP－jadr＝50
機械語：７４５０

アセンブラーのオペランドではとびさき番地 (jadr) を指定するが、機械語ではIPに対する変位を指定する。IPは次の命令の番地を指している。
IPに対するとび先の変位が－128 バイトから＋127 バイト以内の場合にはdisp-Lを指定する。これをこえる場合にはdisp-Lとdisp-Hを指定する。セグメントをまたがる場合は、セグメントベースをseg-L とsrg-H で指定する。

・typeまたはport

OP-CODE	type／port

(例) ＩＮＴ　５
機械語：ＣＤ０５

（４）形式４

オペランドなし。または、レジスターが１バイト目で決まるもの
(機械語コード表のオペランド指定が大文字で表したレジスターだけ)

１バイト目・・・・・オペレーションコード

OP-CODE

(例) ＭＯＶＳＢ
機械語：Ａ４

(例) ＩＮＣ　ＡＸ
機械語：４０

《8086相当機械語コード》　reg／adrはレジスターまたはアドレス。B／Wはバイトまたはワード

OP	ニーモニック	オペランド指定	B/W
00	ADD	reg／adr，reg	B
01	ADD	reg／adr，reg	W
02	ADD	reg，reg／adr	B
03	ADD	reg，reg／adr	W
04	ADD	AL，data	B
05	ADD	AX，data	W
06	PUSH	ES	
07	POP	ES	
08	OR	reg／adr，reg	B
09	OR	reg／adr，reg	W
0A	OR	reg，reg／adr	B
0B	OR	reg，reg／adr	W
0C	OR	AL，data	B
0D	OR	AX，data	W
0E	PUSH	CS	
0F	――――		
10	ADC	reg／adr，reg	B
11	ADC	reg／adr，reg	W
12	ADC	reg，reg／adr	B
13	ADC	reg，reg／adr	W
14	ADC	AL，data	B
15	ADC	AX，data	W
16	PUSH	SS	
17	POP	SS	
18	SBB	reg／adr，reg	B
19	SBB	reg／adr，reg	W
1A	SBB	reg，reg／adr	B
1B	SBB	reg，reg／adr	W

OP	ニーモニック	オペランド指定	B/W
1C	SBB	AL，data	B
1D	SBB	AX，data	W
1E	PUSH	DS	
1F	POP	DS	
20	AND	reg／adr，reg	B
21	AND	reg／adr，reg	W
22	AND	reg，reg／adr	B
23	AND	reg，reg／adr	W
24	AND	AL，data	B
25	AND	AX，data	W
26	ES：		
27	DAA		
28	SUB	reg／adr，reg	B
29	SUB	reg／adr，reg	W
2A	SUB	reg，reg／adr	B
2B	SUB	reg，reg／adr	W
2C	SUB	AL，data	B
2D	SUB	AX，data	W
2E	CS：		
2F	DAS		
30	XOR	reg／adr，reg	B
31	XOR	reg／adr，reg	W
32	XOR	reg，reg／adr	B
33	XOR	reg，reg／adr	W
34	XOR	AL，data	B
35	XOR	AX，data	W
36	SS：		
37	AAA		

reg ／ adrはレジスターまたはアドレス。B／Wはバイトまたはワード

ＯＰ	ニーモニック	オペランド指定	B／W
３８	CMP	reg ／ adr , reg	B
３９	CMP	reg ／ adr , reg	W
３A	CMP	reg , reg ／ adr	B
３B	CMP	reg , reg ／ adr	W
３C	CMP	AL , data	B
３D	CMP	AX , data	W
３E	DS：		
３F	AAS		
４０	INC	AX	W
４１	INC	CX	W
４２	INC	DX	W
４３	INC	BX	W
４４	INC	SP	W
４５	INC	BP	W
４６	INC	SI	W
４７	INC	DI	W
４８	DEC	AX	W
４９	DEC	CX	W
４A	DEC	DX	W
４B	DEC	BX	W
４C	DEC	SP	W
４D	DEC	BP	W
４E	DEC	SI	W
４F	DEC	DI	W
５０	PUSH	AX	W
５１	PUSH	CX	W
５２	PUSH	DX	W
５３	PUSH	BX	W

ＯＰ	ニーモニック	オペランド指定	B／W
５４	PUSH	SP	W
５５	PUSH	BP	W
５６	PUSH	SI	W
５７	PUSH	DI	W
５８	POP	AX	W
５９	POP	CX	W
５A	POP	DX	W
５B	POP	BX	W
５C	POP	SP	W
５D	POP	BP	W
５E	POP	SI	W
５F	POP	DI	W
６０～６F	――――		
７０	JO	jadr	
７１	JNO	jadr	
７２	JB／JNAE	jadr	
７３	JNB ／JAE	jadr	
７４	JE／JZ	jadr	
７５	JNE ／JNZ	jadr	
７６	JBE ／JNA	jadr	
７７	JNBE／JA	jadr	
７８	JS	jadr	
７９	JNS	jadr	
７A	JP／JPE	jadr	
７B	JNP ／JPO	jadr	
７C	JL／JNGE	jadr	
７D	JNL ／JGE	jadr	

reg ／ adrはレジスターまたはアドレス。B／Wはバイトまたはワード

ＯＰ	ニーモニック	オペランド指定	B／W
７Ｅ	JLE ／JNG	jadr	
７Ｆ	JNLE／JG	jadr	
８０	別表①	reg ／ adr，data	B
８１	別表①	reg ／ adr，data	W
８２	別表①	reg ／ adr，data	B
８３	別表①	reg ／ adr，data	W
８４	ＴＥＳＴ	reg ／ adr，reg	B
８５	ＴＥＳＴ	reg ／ adr，reg	W
８６	ＸＣＨＧ	reg ／ adr，reg	B
８７	ＸＣＨＧ	reg ／ adr，reg	W
８８	ＭＯＶ	reg ／ adr，reg	B
８９	ＭＯＶ	reg ／ adr，reg	W
８Ａ	ＭＯＶ	reg ，reg ／ adr	B
８Ｂ	ＭＯＶ	reg ，reg ／ adr	W
８Ｃ	ＭＯＶ	reg ／ adr，seg 注１	W
８Ｄ	ＬＥＡ	reg ， adr	W
８Ｅ	ＭＯＶ	seg ，reg ／ adr 注１	W
８Ｆ	ＰＯＰ	reg ／ adr 注２	W
９０	ＮＯＰ		
９１	ＸＣＨＧ	ＡＸ，ＣＸ	W
９２	ＸＣＨＧ	ＡＸ，ＤＸ	W
９３	ＸＣＨＧ	ＡＸ，ＢＸ	W
９４	ＸＣＨＧ	ＡＸ，ＳＰ	W
９５	ＸＣＨＧ	ＡＸ，ＢＰ	W
９６	ＸＣＨＧ	ＡＸ，ＳＩ	W
９７	ＸＣＨＧ	ＡＸ，ＤＩ	W
９８	ＣＢＷ		
９９	ＣＷＤ		

ＯＰ	ニーモニック	オペランド指定	B／W
９Ａ	ＣＡＬＬ	jadr（セグメント間直接）	
９Ｂ	ＷＡＩＴ		
９Ｃ	PUSHF		
９Ｄ	ＰＯＰＦ		
９Ｅ	ＳＡＨＦ		
９Ｆ	ＬＡＨＦ		
Ａ０	ＭＯＶ	ＡＬ，adr 注３	B
Ａ１	ＭＯＶ	ＡＸ，adr 注３	W
Ａ２	ＭＯＶ	adr ，ＡＬ 注３	B
Ａ３	ＭＯＶ	adr ，ＡＸ 注３	W
Ａ４	MOVSB		B
Ａ５	MOVSW		W
Ａ６	CMPSB		B
Ａ７	CMPSW		W
Ａ８	ＴＥＳＴ	ＡＬ，data	B
Ａ９	ＴＥＳＴ	ＡＸ，data	W
ＡＡ	STOSB		B
ＡＢ	STOSW		W
ＡＣ	LODSB		B
ＡＤ	LODSW		W
ＡＥ	SCASB		B
ＡＦ	SCASW		W
Ｂ０	ＭＯＶ	ＡＬ，data	B
Ｂ１	ＭＯＶ	ＣＬ，data	B
Ｂ２	ＭＯＶ	ＤＬ，data	B
Ｂ３	ＭＯＶ	ＢＬ，data	B
Ｂ４	ＭＯＶ	ＡＨ，data	B
Ｂ５	ＭＯＶ	ＣＨ，data	B

reg ／ adrはレジスターまたはアドレス。B／Wはバイトまたはワード

OP	ニーモニック	オペランド指定	B／W
B6	MOV	DH，data	B
B7	MOV	BH，data	B
B8	MOV	AX，data	W
B9	MOV	CX，data	W
BA	MOV	DX，data	W
BB	MOV	BX，data	W
BC	MOV	SP，data	W
BD	MOV	BP，data	W
BE	MOV	SI，data	W
BF	MOV	DI，data	W
C0	－－－－		
C1	－－－－		
C2	RET	data	W
C3	RET		
C4	LES	reg ，adr	W
C5	LDS	reg ，adr	W
C6	MOV	adr ，data 注2	B
C7	MOV	adr ，data 注2	W
C8	－－－－		
C9	－－－－		
CA	RET	data（セグメント間）	W
CB	RET	（セグメント間）	
CC	INT	3	B
CD	INT	type（≠3）	B
CE	INTO		
CF	IRET		
D0	別表②	reg ／ adr，1	B
D1	別表②	reg ／ adr，1	W

OP	ニーモニック	オペランド指定	B／W
D2	別表②	reg ／ adr，CL	B
D3	別表②	reg ／ adr，CL	W
D4	AAM	（2バイト目＝0A）	
D5	AAD	（2バイト目＝0A）	
D6	－－－－		
D7	XLAT		
D8	ESC	命令，reg ／ adr 注4	
D9	ESC	命令，reg ／ adr 注4	
DA	ESC	命令，reg ／ adr 注4	
DB	ESC	命令，reg ／ adr 注4	
DC	ESC	命令，reg ／ adr 注4	
DD	ESC	命令，reg ／ adr 注4	
DE	ESC	命令，reg ／ adr 注4	
DF	ESC	命令，reg ／ adr 注4	
E0	LOOPNE／LOOPNZ	jadr	
E1	LOOPE ／LOOPZ	jadr	
E2	LOOP	jadr	
E3	JCXZ	jadr	
E4	IN	AL，port	B
E5	IN	AX，port	W
E6	OUT	port，AL	B
E7	OUT	port，AX	W
E8	CALL	jadr	
E9	JMP	jadr	
EA	JMP	jadr（セグメント間直接）	
EB	JMPS	jadr	
EC	IN	AL，DX	B
ED	IN	AX，DX	W

reg ／ adrはレジスターまたはアドレス。B／Wはバイトまたはワード

ＯＰ	ニーモニック	オペランド指定	B／W
ＥＥ	ＯＵＴ	ＤＸ，ＡＬ	B
ＥＦ	ＯＵＴ	ＤＸ，ＡＸ	W
Ｆ０	ＬＯＣＫ		
Ｆ１	――――		
Ｆ２	REPNE ／REPNZ		
Ｆ３	REP ／REPE／REPZ		
Ｆ４	ＨＬＴ		
Ｆ５	ＣＭＣ		
Ｆ６	別表③	reg ／ adr	B

ＯＰ	ニーモニック	オペランド指定	B／W
Ｆ７	別表③	reg ／ adr	W
Ｆ８	ＣＬＣ		
Ｆ９	ＳＴＣ		
ＦＡ	ＣＬＩ		
ＦＢ	ＳＴＩ		
ＦＣ	ＣＬＤ		
ＦＤ	ＳＴＤ		
ＦＥ	別表④	reg ／ adr	B
ＦＦ	別表④	reg ／ adr	W

注１　形式１で２バイト目のb5＝0,b4～b3＝セグメントレジスターとして指定
注２　形式２で xxxは 000のみ
注３　アドレスの直接指定。２バイト目に番地の下位バイト、３バイト目に上位バイト
注４　形式２と同じだが、１バイト目の下位３ビットと２バイト目のb5～b3（xxx ）の３ビットの計６ビットで命令。ｍｏｄ，ｒｅｇ，ｄｉｓｐでレジスターやメモリーを指定

別表①～④は形式２に属する

《別表①》　　８０～８３　ニーモニック　　　　オペランド指定　reg ／ adr，data

ＯＰ	xxx　（２バイト目　mod　，xxx　，reg ／ adr）							
	000	001	010	011	100	101	110	111
８０～８１	ＡＤＤ	ＯＲ	ＡＤＣ	ＳＢＢ	ＡＮＤ	ＳＵＢ	ＸＯＲ	ＣＭＰ
８２～８３	ＡＤＤ	―――	ＡＤＣ	ＳＢＢ	―――	ＳＵＢ	―――	ＣＭＰ

OP＝82は80と同じ。OP＝83はdataを１バイトにして符号付き拡張

《別表②》　　Ｄ０～Ｄ３　ニーモニック　　　　オペランド指定　reg ／ adr，１またはＣＬ

ＯＰ	xxx　（２バイト目　mod　，xxx　，reg ／ adr）							
	000	001	010	011	100	101	110	111
Ｄ０～Ｄ３	ＲＯＬ	ＲＯＲ	ＲＣＬ	ＲＣＲ	ＳＡＬ ＳＨＬ	ＳＨＲ	―――	ＳＡＲ

《別表③》　　Ｆ６～Ｆ７　ニーモニック　　　　オペランド指定　reg ／ adr

OP	xxx （2バイト目 mod ,xxx ,reg ／ adr)							
	000	001	010	011	100	101	110	111
Ｆ６～Ｆ７	ＴＥＳＴ	―――	ＮＯＴ	ＮＥＧ	ＭＵＬ	ＩＭＵＬ	ＤＩＶ	ＩＤＩＶ

ＴＥＳＴのみオペランド指定が reg／adr ,dataとなる

《別表④》　　ＦＥ～ＦＦ　ニーモニック　　　　オペランド指定　reg ／ adr

OP	xxx （2バイト目 mod ,xxx ,reg ／ adr)							
	000	001	010	011	100	101	110	111
ＦＥ	ＩＮＣ	ＤＥＣ	―――	―――	―――	―――	―――	―――
ＦＦ	ＩＮＣ	ＤＥＣ	ＣＡＬＬ	ＣＡＬＬ	ＪＭＰ	ＪＭＰ	ＰＵＳＨ	―――

xxx ＝010,100 はセグメント内間接CALL,JMP。reg またはadr の内容が示す番地にとぶ
xxx ＝011,101 はセグメント間間接CALL,JMP。オペランドはadr のみ。指定したメモリー番地の内容がとびさき番地。とび先のコードセグメント番地は、（メモリー番地＋２）の２バイトで指定する

10．9801　USR関数とCALL文の使い方

☆機械語プログラムの作成とロード☆

《機械語プログラムの作成 —— DISK－BASICの場合》

（1）　CLEAR文を使って機械語プログラム領域をメモリーに確保する。
（2）　実際にどこに機械語プログラムを作るかを決める。ここで決めたセグメントベース番地とオフセットは、BASICで機械語プログラムを使うときに必要なので必ず記録しておく。
（3）　モニターに入り、Cコマンドを使って機械語セグメントベース番地を（2）で決めた番地に設定する。モニターに入る前にDEF　SEG文を実行してもよい。
（4）　Aコマンドを使って（2）で決めたオフセットから機械語プログラムを入力できるようにする。
（5）　アセンブラーニーモニックを使って、機械語プログラムを入力する。
（6）　BASICに戻り、BSAVE命令で機械語プログラムをディスクに保存する。

(例)
```
CLEAR ,&H1F00                     機械語プログラム領域の確保
OK
MON                               モニターを動かす
h〕C1F00                          機械語セグメントベース番地の設定
h〕A0000                          オフセットがゼロの場所からプログラムを入力する
0000  2E8C160001  MOV  CS:〔0100〕,SS  ┐ ニーモニックで機械語
0005  2E89260201  MOV  CS:〔0102〕,SP  ┘ プログラムを入れる
            :
h〕^B                             BASICに戻る
OK
BSAVE"PROG0",&H0000,&H0400
                                  機械語プログラムをディスクに保存する
```

上の例では、機械語エリアのセグメントベース番地として1F00Hを指定している。RAMが128KBの場合、RAMの番地は00000Hから1FFFFHにあるから、機械語エリアとして絶対番地で1F000Hから1FFFFHまでの1000H＝4KBが取られる。

《機械語プログラムの作成 —— ROM－BASICの場合》

ディスクがないとモニターのAコマンドは使えない。手作業でプログラムを機械語におきかえ、Eコマンドを使って16進数で入力する。BSAVE命令は使えないので、モニターのWコマンドを使って機械語プログラムをカセットに保存する。

(例)
```
CLEAR ,&H1F00                     機械語プログラム領域の確保
OK
MON                               モニターを動かす
h〕C1F00                          機械語セグメントベース番地の設定
h〕E0000                          オフセットがゼロの場所からプログラムを入力する
0000  2E 8C 16 00 01 2E 89 26 02 01 ..........
            :
            :                     16進数でプログラムを入れる
h〕WPROG0,0000,0400
h〕^B
OK
```

《ディスクに保存したプログラムのロード》

つぎのような手順で、メモリーにロードする。

（1）　CLEAR文を使って機械語プログラム領域を確保する。
（2）　DEF　SEG文を使って、機械語セグメントベース番地をプログラムを作った時と同じにする
（3）　BLOAD命令を使って、機械語プログラムをメモリーにロードする。

BASICの命令だけを使って準備作業ができるので、プログラムの先頭で次のような処理する。

(例)
```
10 CLEAR ,&H1F00            機械語プログラム領域の確保
20 DEF SEG=&H1F00           機械語セグメントベース番地の設定
30 BLOAD "PROG0"            機械語プログラムをロードする
```

《カセットに保存したプログラムのロード》
モニターのRコマンドを使ってメモリーにロードする。この場合、ＢＡＳＩＣの命令だけでは準備作業はできない。ＢＡＳＩＣのプログラムを実行する前に次のような操作をする。

```
(例) CLEAR ,&H1F00            機械語プログラム領域の確保
     OK
     DEF SEG=&H1F00           機械語セグメントベース番地の設定
     OK
     MON                      モニターを動かす
     h〕RPROG0                機械語プログラムをロードする
     Found:PROG0
     h〕^B                    ＢＡＳＩＣに戻る
     OK
```

《ＰＯＫＥ文で機械語を扱う方法》
モニターの機能を使わず、ＰＯＫＥ文を使い１６進数の形で機械語プログラムをメモリーに入れることができる。次のような処理をプログラムの先頭に書いておく。

```
(例) 10 CLEAR ,&H1F00             機械語プログラム領域の確保
     20 DEF SEG=&H1F00            機械語セグメントベース番地の設定
     40 POKE &H0000,&H2E          １６進数の形で機械語プログラムを入れる
     50 POKE &H0001,&H8C
              ⋮
```

ＤＡＴＡ文を使ってプログラム入力する方法は２０１頁参照。

☆機械語プログラムの呼び出し方☆

機械語で書いたプログラムをＢＡＳＩＣのプログラムで利用するには、ＵＳＲ関数（ユーザー関数）として呼び出す方法と、ＣＡＬＬ文でサブルーチンとして呼び出す方法の二通りある。

《ＵＳＲ関数》
ＵＳＲ関数で機械語のプログラムを呼び出すと、ＢＡＳＩＣの関数と同じように使える。計算の結果はＵＳＲ関数自体が持つので、計算式の中でＵＳＲ関数を使ったり、ＩＦ文の条件に書くなど、さまざまに使える。ＵＳＲ関数に返す値の型は、引き数の型と同じにする。引き数には定数、変数のほか、式が書ける。ただし、引き数は一つしか書けない。ＵＳＲ関数はＤＥＦ　ＵＳＲ文で先頭番地を宣言してから呼び出す。

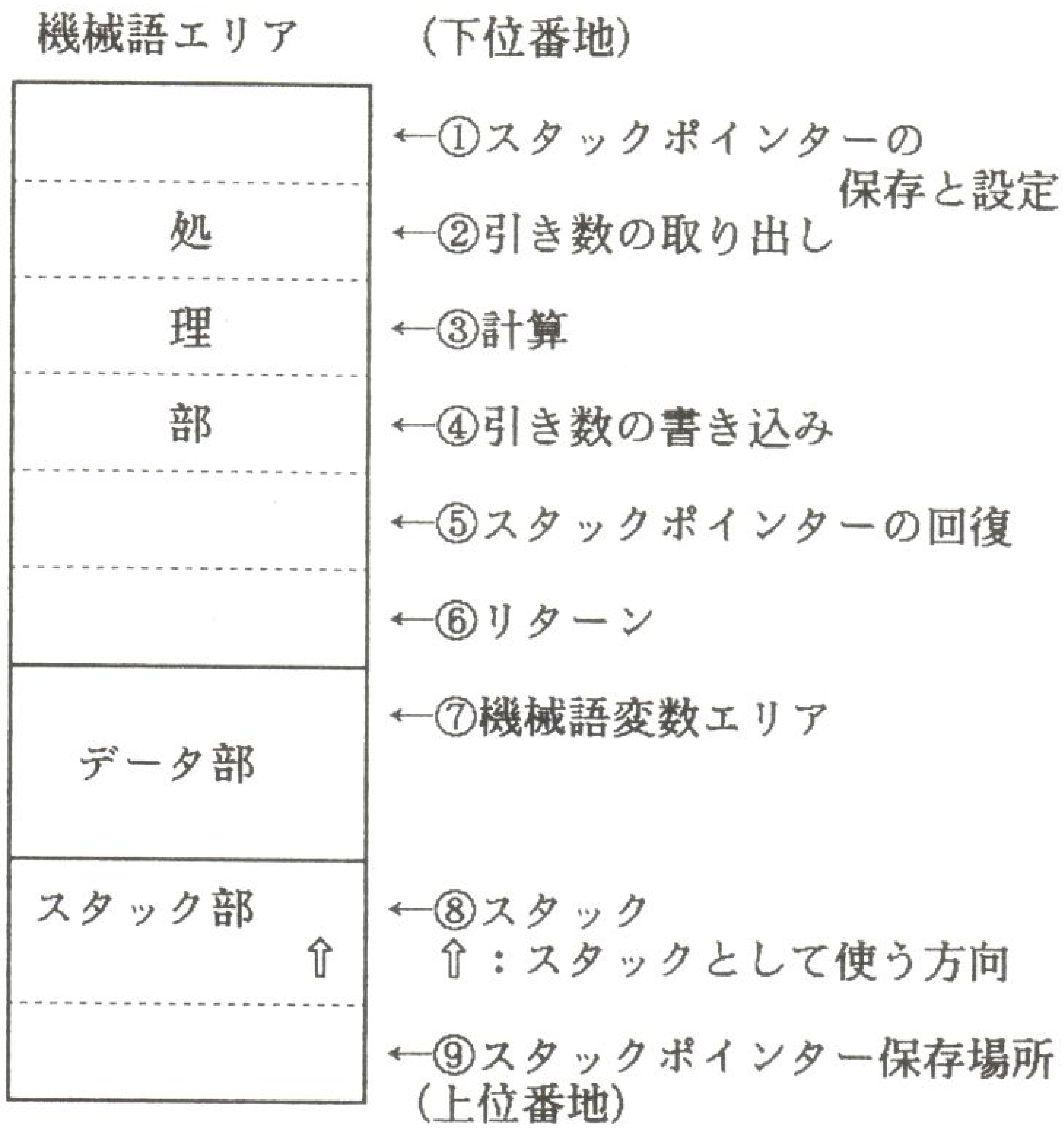
```
(例) 10   CLEAR ,&H1F00          機械語プログラム領域の確保
     20   DEF SEG=&H1F00         機械語セグメントベース番地の設定
     30   BLOAD "PROG0"          機械語プログラムをロードする
     100  DEF USR0=&H0           ＵＳＲ関数を先頭番地（オフセット）の宣言
     110  INPUT A
     120  B=USR0(A)              ＵＳＲ関数の呼び出す
     130  PRINT A,B
     140  END
```

《ＣＡＬＬ文》
ＣＡＬＬ文で機械語プログラム（機械語サブルーチン）を呼び出して使うときは、ＵＳＲ関数とは異なり複数の引き数を受け渡しできる。引き数は機械語サブルーチンに値を渡すだけではなく、値を返すためにも使える。ただし、引き数には変数しか書けない。機械語サブルーチンを呼び出すには、先頭番地をサブルーチン名（変数）に入れておく必要がある。

```
(例) 10   CLEAR ,&H1F00          機械語プログラム領域の確保
     20   DEF SEG=&H1F00         機械語セグメントベース番地の設定
     30   BLOAD "PROG1"          機械語プログラムをロードする
     100  SUB1=&H0000            サブルーチン名に先頭番地（オフセット）を入れる
     110  INPUT A
     120  CALL SUB1(A,B)         機械語プログラムを呼び出す
     130  PRINT A,B
     140  END
```

☆機械語プログラムの作り方☆

《機械語プログラムの構造》
機械語プログラムは、右図の①～⑥の処理部と⑦～⑨のデータ部とスタック部から成る。機械語プログラムがＵＳＲ関数やＣＡＬＬ文から呼ばれたとき、ＤＥＦ　ＳＥＧ文で指定した機械語セグメントベース番地がＣＳレジスターに入る。
ＵＳＲ関数とＣＡＬＬ文の「入り口の処理」ではスタックポインターを保存してから設定する。また、ＵＳＲ関数やＣＡＬＬ文に戻るときの「出口の処理」ではスタックポインターを元に戻す。
引き数を取り出したり書き込んだりするとき、ＵＳＲ関数とＣＡＬＬ文では引き数の扱い方がちがう。また、引き数が数値か文字列かによっても引き数の扱い方が異なる。



《入り口と出口の処理》
機械語プログラムからＢＡＳＩＣに戻るには必ずＩＲＥＴ命令を使う。ＢＡＳＩＣは機械語プログラムを機械語のＩＮＴ命令で呼び出しているため、スタックには戻り番地としてオフセット番地とベース番地の４バイトが入っている。
ＢＡＳＩＣに戻るときには必ずスタックポインターは呼び出されたときと同じ番地を指すようにする。

《スタックの処理》
スタックを８レベル（１６バイト）を超えて使うときは、機械語エリアにスタック部を用意する。スタック部の最後の４バイトをスタックポインターの保存場所として、それ以前をスタックとして使う。スタックが８レベル以下のときは、スタック部を用意しなくてよい。
機械語プログラムに入ったとき、ＤＳレジスターは引き数のセグメントベース番地を、ＣＳレジスターは機械語エリアのセグメントベース番地を指している。そこで、スタックポインターの処理では、ＣＳレジスターでオペランドのセグメントベース番地を指定する。

スタックポインターの保存と設定（nnnnは⑨スタックポインター保存場所）

```
MOV  CS:〔nnnn〕,SS        SSレジスターの値を〔nnnn〕番地にしまう
MOV  CS:〔nnnn+2〕,SP      SPレジスターの値を〔nnnn+2〕番地にしまう
MOV  AX,CS                 機械語セグメントの場所をAXに取り出す
MOV  SS,AX                 取り出した値をSSにセットする
MOV  SP,nnnn               機械語セグメントの中でのスタックの場所をSPにセットする
```

スタックポインターの回復

```
MOV  SS,CS:〔nnnn〕        SSの回復
MOV  SP,CS:〔nnnn+2〕      SPの回復
```

☆引き数の取り出しと書き込み☆

《ＵＳＲ関数を使う場合の引き数の扱い》
機械語プログラムがＵＳＲ関数から呼ばれたときは、引き数の型がＡレジスターに入り、引き数がＦＡＣ（浮動小数点アキュムレーター）に入る。ＦＡＣはＢＡＳＩＣが使う８バイトの作業場所で、ＤＳレジスターがＦＡＣのセグメント番地、ＢＸレジスターがＦＡＣのオフセットを指す。
引き数が整数型か単精度実数型か倍精度実数型か文字列型かによってＦＡＣの各バイトの働きが異なる。また、ＢＸレジスターが指すＦＡＣのオフセットもちがう。
機械語プログラムでは、ＡＬレジスターに引き数の型を、ＦＡＣに計算結果をセットして、ＵＳＲ関数に値を返す。
ＵＳＲ関数と機械語プログラムで引き数をやりとりするときの、ＡＬレジスター・ＤＳレジスター・ＢＸレジスター・ＦＡＣの働きをまとめたのが、つぎの表である。

引き数の型	整数型	単精度実数型	倍精度実数型	文　字　列　型
ＤＳレジスターの値	ＦＡＣのセグメントベース番地			
ＢＸレジスターの値	ＦＡＣの５バイト目のオフセット		ＦＡＣの１バイト目のオフセット	ＦＡＣの５バイト目のオフセット
ＡＬレジスターの値	2	4	8	3
ＦＡＣの１バイト目			仮数の最下位８bit	
ＦＡＣの２バイト目			仮数の第６位８bit	
ＦＡＣの３バイト目			仮数の第５位８bit	
ＦＡＣの４バイト目			仮数の第４位８bit	
ＦＡＣの５バイト目	下位８bit	仮数の下位８bit	仮数の第３位８bit	sdオフセット下位バイト
ＦＡＣの６バイト目	上位８bit	仮数の中位８bit	仮数の第２位８bit	sdオフセット上位バイト
ＦＡＣの７バイト目		仮数の上位８bit	仮数の最上位８bit	sdベース下位バイト
ＦＡＣの８バイト目		指数部	指数部	sdベース上位バイト

ベース：セグメントベース番地、ｓｄ：ストリングディスクリプター

（1）引き数が数値の場合
ＵＳＲ関数からＡＬレジスターに引き数の型、ＤＳレジスターとＢＸレジスターにＦＡＣの場所が渡される。ＦＡＣ（浮動小数点アキュムレーター）には引き数が入っている。

（例）引き数が整数のとき、ＵＳＲ関数から呼ばれる機械語プログラムは次のように作る。

ＦＡＣの場所を保存する（nnnnは⑦機械語変数エリア）
```
MOV  CS:〔nnnn〕,BX        FACのオフセットとセグメントベース番地を
MOV  CS:〔nnnn＋2〕,DS     保存する
```

数値をＦＡＣから取り出す
```
MOV  AX,〔BX〕             BXをオフセットに使い、値をAXに取り出す
```

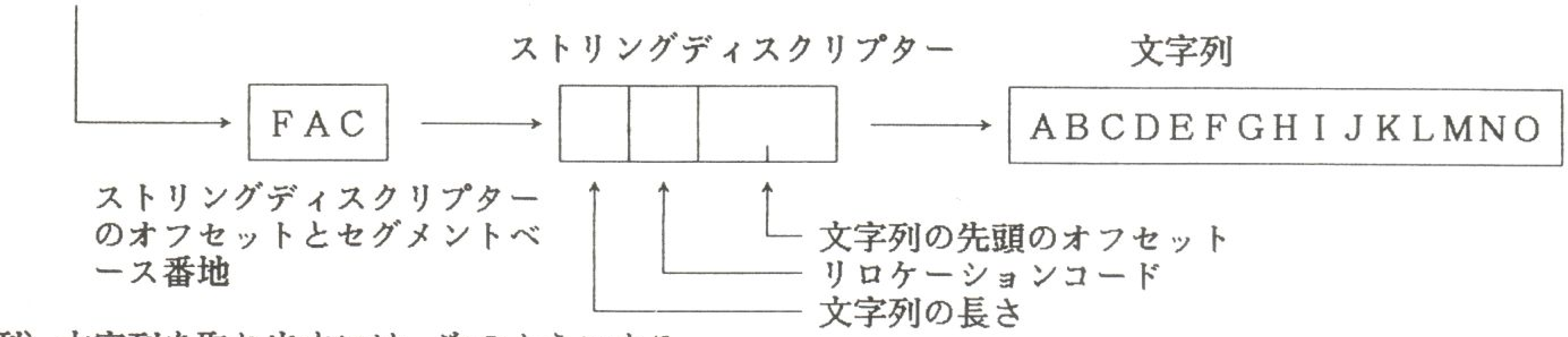
取り出した値を使い計算する、結果はＡＸに入れる

ＡＸに入っている計算結果をＦＡＣに入れる
```
LDS  BX,CS:〔nnnn〕        保存しておいたFACの番地をDSとBXに入れ
MOV  〔BX〕,AX             計算結果をFACに入れる
```

ＢＡＳＩＣのプログラムに戻る
```
IRET
```

（2）引き数が文字列の場合
機械語プログラムがＵＳＲ関数から呼ばれたときは、引き数の型（＝3）がＡＬレジスターに入り、ＦＡＣの場所がＤＳレジスターとＢＸレジスターに渡される。ＦＡＣにストリングディスクリプターのオフセットとセグメントベース番地が入っている。ストリングディスクリプターは４バイトで、引き数の文字列の長さと番地を示す。引き数の番地はリロケーションコードとオフセットで表す。リロケーションコードが００Ｈなら文字列はＤＸレジスターの示すシンボルテーブルセグメントにあり、００Ｈ以外ならテキストセグメント（セグメントベース番地は００６０Ｈ）にある。機械語プログラムの中でストリングディスクリプターの内容を変えてはいけない。

〔DS〕：FACのセグメントベース番地
〔BX〕：FACのオフセット



(例) 文字列を取り出すには、次のようにする。

文字列の場所をDS・SIに、文字列の長さをCXに入れる

```
         LDS  BX,〔BX〕      FACからDSとBXにストリングディスクリプターの場所
                            を取り出す
         MOV  CX,〔BX〕      ストリングディスクリプターを取り出す
                            CHにリロケーションコード、CLに長さがそれぞれ入る
         CMP  CH,00         リロケーションコードの値にしたがって文字列のセグメント
                            ベース番地を設定する
         JZ   XXXX番地       ゼロならDXの値を使う

         XOR  CH,CH         CXを文字列の長さとして使うため、CHをゼロにする

         MOV  DX,0060       文字列は0060Hから始まるセグメントにあるので、DXを
                            設定しなおす
XXXX番地 MOV  SI,2〔BX〕     SIにオフセットを入れる

         MOV  DS,DX         DSにセグメントベース番地を入れる
```

文字列から文字を取り出すにはつぎのようにする

```
         MOV  AL,〔SI〕      ALに文字列の先頭の1文字を取り出す
```

文字列を置き換えるには、次のようにする

```
         MOV  〔SI〕,AL      文字列の先頭1文字にALの内容を入れる
```

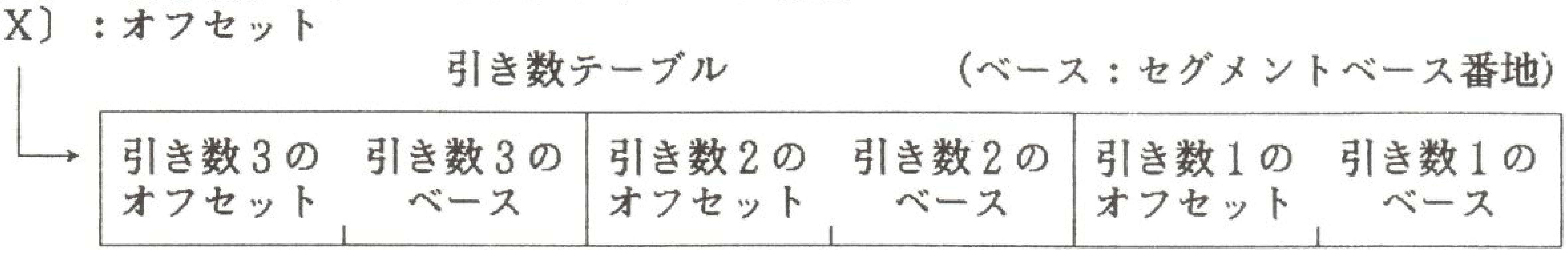
《CALL文を使う場合の引き数の扱い》

機械語プログラムがCALL文から呼ばれたときは、引き数の番地がテーブルで渡される。このテーブルを引き数テーブルという。DSレジスターとBXレジスターが引き数テーブルの場所を示す。DSレジスターが引き数テーブルのセグメントベース番地を、BXレジスターがオフセットを持つ。

引き数テーブルには引き数の並びかたの逆順に、それぞれのセグメントベース番地とオフセットが入っている。引き数が数値の場合、整数型（2バイト）、単精度実数型（4バイト）、倍精度実数型（8バイト）の先頭番地を指す。文字列の場合は、ストリングディスクリプターの先頭番地を指す。

機械語プログラムからCALL文に結果を返すときは、直接引き数に値を入れる。

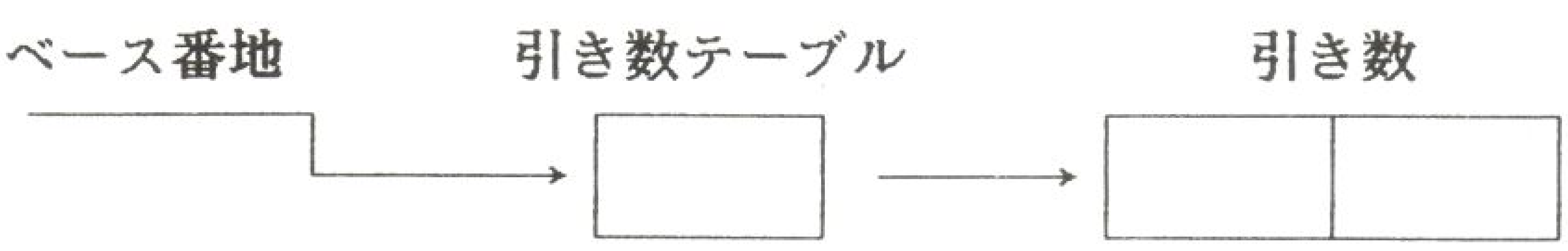
CALL　機械語プログラム（引き数1，引き数2，引き数3）
〔DS〕：引き数テーブルのセグメントベース番地
〔BX〕：オフセット



(1) 引き数が数値の場合

引き数テーブルに、引き数のある先頭番地のセグメントベース番地とオフセットが入っている。

〔DS〕：引き数テーブルのセグメントベース番地
〔BX〕：引き数テーブルのオフセット



（例）ＣＡＬＬ　ＳＵＢ（Ａ％，Ｂ％，Ｃ％，Ｄ％）の場合、４番目の引き数Ｄ％の値をＡＸレジスターに取り出すには次のようにする

Ｄ％は最後の引き数だから引き数テーブルの１番目に番地が入っている。

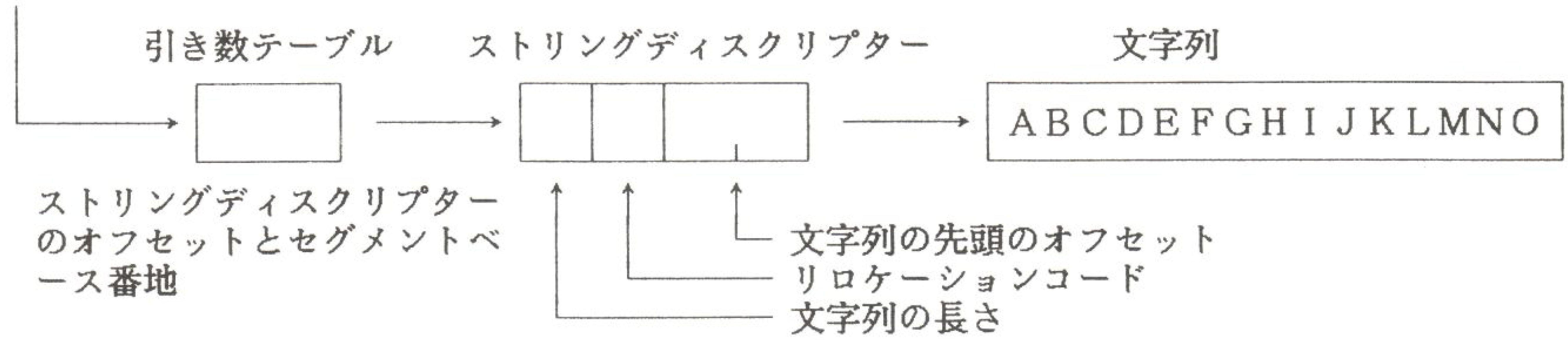
また、ＤＳレジスターとＢＸレジスターは引き数テーブルの場所を示しているため、引き数の取り出しが終わるまで変えられない。オペランドの番地指定にはＥＳレジスターとＳＩレジスターを使う。

ＬＥＳ　ＳＩ，０〔ＢＸ〕	ＢＸは引き数テーブルのオフセット Ｄ％は引き数テーブルの１番目なので、０〔ＢＸ〕とする。２番目なら４〔ＢＸ〕と、４の倍数になる。 この命令を実行するとＥＳに引き数のセグメントベース番地、ＳＩにオフセットが入る
ＭＯＶ　ＡＸ，ＥＳ：〔ＳＩ〕	ＡＸに引き数の内容を取り出す ＥＳ：はセグメントプレフィックスと呼び、この命令の対象のセグメントをＤＳからＥＳに切り換える

（２）引き数が文字列の場合

引き数テーブルによって指された引き数のストリングディスクリプターには、文字列の長さと場所が入っている。文字列の場所はリロケーションコードとオフセットで表す。リロケーションコードが００Ｈならば文字列はストリングディスクリプターと同じセグメントにあり、００Ｈ以外ならテキストセグメント（セグメントベース番地は００６０Ｈ）にある。

〔ＤＳ〕：引き数テーブルのセグメントベース番地
〔ＢＸ〕：引き数テーブルのオフセット



（例）ＣＡＬＬ　ＳＵＢ（Ａ＄，Ｂ＄，Ｃ＄，Ｄ＄）の場合、Ｄ＄の最初の１文字を取り出すには次のようにする。

Ｄ＄は最後の引き数だから引き数テーブルの１番目に番地が入っている。

文字列の場所をＥＳ・ＳＩに、文字列の長さをＣＸに入れる

	ＬＥＳ　ＳＩ，０〔ＢＸ〕	ＥＳとＳＩにＤ＄のストリングディスクリプターの番地を入れる
	ＭＯＶ　ＣＸ，ＥＳ：〔ＳＩ〕	ストリングディスクリフターを取り出す ＣＨにリロケーションコード、ＣＬに長さがそれぞれ入る
	ＭＯＶ　ＤＸ，ＥＳ	ストリングディスクリプターのあるセグメントベース番地をＤＸに入れる
	ＣＭＰ　ＣＨ，００	リロケーションコードの値に従って文字列のセグメントベース番地を設定する
	ＪＺ　ＸＸＸＸ番地	ゼロなら文字列はストリングディスクリプターと同じセグメントにある
	ＸＯＲ　ＣＨ，ＣＨ	ＣＸを文字列の長さとして使うためＣＨをゼロにする
	ＭＯＶ　ＤＸ，００６０	文字列は００６０Ｈから始まるセグメントにあるのでＤＸを設定しなおす
ＸＸＸＸ番地	ＭＯＶ　ＳＩ，ＥＳ：２〔ＳＩ〕	ＳＩにオフセットを入れる
	ＭＯＶ　ＥＳ，ＤＸ	ＥＳにセグメントベース番地を入れる

文字列から文字を取り出すにはつぎのようにする

ＭＯＶ　ＡＬ，ＥＳ：〔ＳＩ〕	ＡＬに文字列の先頭１文字を取り出す

文字列を置き換えるには次のようにする。

ＭＯＶ　ＥＳ：〔ＳＩ〕，ＡＬ	文字列の先頭１文字にＡＬの内容を入れる

11．9801　機械語プログラム集

機械語プログラムはモニターまたはＰＯＫＥ命令でメモリー上に作るか、すでにセーブしてあるものを、ＢＬＯＡＤ命令でロードする。詳しくは「ＵＳＲ関数とＣＡＬＬ文の使い方」参照。

☆（９８）　機械語　例題１　☆

１．機能

整数型引き数をビットパターンとして扱い、値が１のビット数を数える。

２．呼び出し形式

ＤＥＦ　ＳＥＧ＝＆Ｈ１Ｆ００
ＤＥＦ　ＵＳＲ１＝＆Ｈ０
ＰＲＩＮＴ　ＵＳＲ１（整数型引き数）

３．利用例

－３から３までの整数の１になっているビット数を数え、ビット数が奇数か偶数かを表示する。

```
100 DEF SEG=&H1F00: DEF USR1=&H0
110 DIM P$(1): P$(0)="ｸﾞｳｽｳ": P$(1)="ｷｽｳ"
120 FOR I%=-3 TO 3
130   PRINT USING "### ## &   &";I%,USR1(I%),P$(USR1(I%) MOD 2)
140 NEXT I%
run
 -3 15 ｷｽｳ
 -2 15 ｷｽｳ
 -1 16 ｸﾞｳｽｳ
  0  0 ｸﾞｳｽｳ
  1  1 ｷｽｳ
  2  1 ｷｽｳ
  3  2 ｸﾞｳｽｳ
```

４．制約事項

整数型以外の引き数ではなにもしない。このときＵＳＲ１の値は引き数の値と同じになる。

５．機械語プログラム　　入り口状態　：ＡＬ－　引き数の型番号（整数型ならば２）
ＢＸ－　ＦＡＣの整数型引き数の値の下位１バイトのオフセット
ＤＳ－　ＦＡＣのセグメントベース
ＣＳ－　コードセグメントベース（値１Ｆ００Ｈ）

```
CS=1F00
0000 3C02      CMP   AL,02      引き数の型が整数型（型番号が２）かどうか調べる
0002 7512      JNE   0016         整数型でなければ戻る
0004 8B07      MOV   AX,[BX]    ＡＸに引き数の値を取り出す
0006 B91000    MOV   CX,0010    ＣＸに繰り返しの回数１６（ビット数）を入れる
0009 31D2      XOR   DX,DX      １になっているビット数のカウンターＤＸをクリアーする
000B D1E0      SAL   AX,1       ＡＸを１ビット左にずらし最上位ビットをＣフラグに入れる
000D 7301      JAE   0010         Ｃフラグがゼロなら数えない
000F 42        INC   DX           Ｃフラグが１ならカウンターＤＸに１をたす
0010 E2F9      LOOP  000B       未処理ビット数ＣＸから１を引きゼロでなければ繰り返す
0012 8917      MOV   [BX],DX    １になっているビット数をＦＡＣに入れる
0014 B002      MOV   AL,02      ＡＬにＵＳＲ関数の返す値の型番号（整数型は２）を入れる
0016 CF        IRET             呼び出しプログラムに戻る
```

☆（98）機械語　例題2　☆

1．機能

文字列型の引き数をもらい、その文字列の中でプリンターに印字できない文字（文字コード００Ｈ～１ＦＨ・７ＦＨ・Ｆ８Ｈ～ＦＦＨ）を「・」（文字コードＡ５Ｈ）に置き換える。

2．呼び出し形式

```
DEF  SEG=&H1F00
DEF  USR2=&H40
PRINT  USR2（文字列型引き数）
```

3．利用例

文字コード順に並んだ文字の表を印字する。

```
100 DEF SEG=&H1F00: DEF USR2=&H40
110 FOR J=0 TO &HF
120  S$=""
130  FOR I=0 TO &HF0 STEP &H10
140   S$=S$+CHR$(I+J)+" "
150  NEXT I
160  LPRINT USR2(S$)
170 NEXT J
run
· ·   0 @ P ` p ▁ ┴   ー タ ミ ═ ╳
· · ! 1 A Q a q ▂ ┬ 。 ア チ ム ╞ 円
· · " 2 B R b r ▃ ┤ 「 イ ツ メ ╪ 年
· · # 3 C S c s ▄ ├ 」 ウ テ モ ╡ 月
· · $ 4 D T d t ▅ ▔ 、 エ ト ヤ ◢ 日
· · % 5 E U e u ▆ ─ ・ オ ナ ユ ◣ 時
· · & 6 F V f v ▇ │ ヲ カ ニ ヨ ◥ 分
· · ' 7 G W g w █ ▕ ァ キ ヌ ラ ◤ 秒
· · ( 8 H X h x ▏ ┌ ィ ク ネ リ ♠ ·
· · ) 9 I Y i y ▎ ┐ ゥ ケ ノ ル ♥ ·
· · * : J Z j z ▍ └ ェ コ ハ レ ♦ ·
· · + ; K [ k { ▌ ┘ ォ サ ヒ ロ ♣ ·
· · , < L ¥ l ¦ ▋ ╭ ャ シ フ ワ ● ·
· · - = M ] m } ▊ ╮ ュ ス ヘ ン ○ ·
· · . > N ^ n ~ ▉ ╰ ョ セ ホ ゛ ／ ·
· · / ? O _ o · ┼ ╯ ッ ソ マ ゜ ＼ ·
```

4．制約事項

文字列型以外の引き数ではなにもしない。このときＵＳＲ２の値は引き数の値と同じになる。

5．機械語プログラム

入り口状態　：ＡＬ－　引き数の型番号（文字列型ならば３）
ＢＸ－　ＦＡＣのストリングディスクリプター（ｓｄ）のオフセット
ＤＸ－　引き数文字列のセグメントベース
ＤＳ－　ＦＡＣのセグメントベース
ＣＳ－　コードセグメントベース（値１Ｆ００Ｈ）

```
CS=1F00
0040 3C03     CMP  AL,03        引き数の型が文字列型（型番号が3）かどうか調べる
0042 7532     JNE  0076            文字列型でなければ戻る
0044 C51F     LDS  BX,[BX]      DS・BXにストリングディスクリプターの番地を取り出す
0046 8B0F     MOV  CX,[BX]      CXに文字列の長さとリロケーションコードを取り出す
0048 80F900   CMP  CL,00        引き数文字列の長さを調べる
004B 7429     JE   0076            引き数文字列の長さがゼロならば戻る
004D 80FD00   CMP  CH,00        リロケーションコードからセグメントベースを求める
0050 7405     JE   0057            CHがゼロならDXの値をセグメントベースとする
0052 30ED     XOR  CH,CH           ゼロでなければCHをクリアーする
0054 BA6000   MOV  DX,0060         セグメントベースとして60Hを設定する
0057 8B7702   MOV  SI,02[BX]    SIに引き数文字列の先頭番地のオフセットを入れる
005A 8EDA     MOV  DS,DX        DSに引き数文字列のセグメントベースを移す
005C 8A04     MOV  AL,[SI]      ALに引き数文字列の1文字を取り出す
```

```
005E 3C20     CMP    AL,20          文字コード２０Ｈと比べる
0060 720C     JB     006E           ２０Ｈ未満なら置き換え処理へ行く
0062 3C7F     CMP    AL,7F          文字コード７ＦＨと比べる
0064 720B     JB     0071           ７ＦＨ未満なら置き換え処理をとばす
0066 3C80     CMP    AL,80          文字コード８０Ｈと比べる
0068 7204     JB     006E           ８０Ｈ未満なら置き換え処理へ行く
006A 3CF8     CMP    AL,F8          文字コードＦ８Ｈと比べる
006C 7203     JB     0071           Ｆ８Ｈ未満なら置き換え処理をとばす
006E C604A5   MOV    BYTE [SI],A5 置き換え処理（ＳＩの指す文字を「・」Ａ５Ｈにする）
0071 46       INC    SI             ＳＩを引き数文字列の次の文字のオフセットにする
0072 E2E8     LOOP   005C         残り文字数ＣＸから１を引きゼロになるまで繰り返す
0074 B003     MOV    AL,03        ＡＬにＵＳＲ関数の返す値の型番号（文字列型３）を入れる
0076 CF       IRET                呼び出しプログラムに戻る
```

☆（９８）機械語　例題３　☆

1．機能

１６進数の数字（０〜９・Ａ〜Ｆまたはａ〜ｆ）から成る文字列を内部形（２進数表現）に変換する。１６進数の数字２文字（２バイト）が内部形１バイトに相当する。

2．呼び出し形式

ＤＥＦ　ＳＥＧ＝＆Ｈ１Ｆ００
ＨＥＸＶＡＬ＝＆Ｈ８０
ＣＡＬＬ　ＨＥＸＶＡＬ（文字列変数１，文字列変数２）
　文字列変数１（入力）　：１６進数の数字から成る文字列
　文字列変数２（出力）　：変換した値を入れる文字列（文字列変数１の半分の長さであること）

3．利用例

プリンターに漢字かな混じりの文を印字する。

```
100 DEF SEG=&H1F00: HEXVAL=&H80
110 H$="1b4b4c5c1b481b26c6d4bbbcb21b4b442b467c3f374a391b481b24"
120 V$=SPACE$(LEN(H$)/2)
130 CALL HEXVAL(H$,V$)
140 LPRINT V$
run
目にやさしい朝日新聞
```

4．制約事項

・引き数の数や型が上の呼び出し形式と違う場合、正しく動かない。
・文字列変数２の長さが文字列変数１の長さの半分でない場合はなにもしない。
・文字列変数１の中に１６進数の数字以外の文字がある場合には、変換した結果は正しくない。

5．機械語プログラム　　入り口状態　：ＢＸ－　引き数テーブルのオフセット
　ＤＳ－　引き数テーブルのセグメントベース
　ＣＳ－　コードセグメントベース（値１Ｆ００Ｈ）
　引き数テーブルー　引き数２、引き数１の順でそれぞれのストリングディスクリプター（ｓｄ２・ｓｄ１）のオフセット・セグメントベースが入っている

```
CS=1F00
0080 C437     LES    SI,[BX]       引き数２の取り出し；ＥＳ・ＳＩにｓｄ２の番地を入れる
0082 26       ES:                    (次の命令はエクストラセグメントベースを使う)
0083 8B0C     MOV    CX,[SI]       ＣＸに引き数２の長さとリロケーションコードを入れる
0085 80F900   CMP    CL,00            引き数２の長さＣＬがゼロかどうか調べる
0088 744E     JE     00D8             引き数２の長さがゼロなら戻る
008A 88C8     MOV    AL,CL            ＡＬに引き数２の長さを保存する
008C 8CC2     MOV    DX,ES            ＤＸにｓｄ２のセグメントベースを入れる
008E 80FD00   CMP    CH,00         引き数２の文字列のセグメントベースを求める
0091 7405     JE     0098             ＣＨがゼロならＤＸの値をセグメントベースとする
```

```
0093 30ED     XOR    CH,CH          ゼロでなければＣＨをクリアーする
0095 BA6000   MOV    DX,0060        ＤＸにセグメントベースとして６０Ｈを設定する
0098 26       ES:                   (次の命令はエクストラセグメントベースを使う)
0099 8B7C02   MOV    DI,02[SI]   ＤＩに引き数２の文字列の先頭番地のオフセットを入れる
009C 8EC2     MOV    ES,DX       ＥＳに引き数２の文字列のセグメントベースを入れる
009E 57       PUSH   DI             引き数２の文字列の先頭番地のオフセットＤＩをしまう
009F 06       PUSH   ES             引き数２の文字列のセグメントベースＥＳをしまう
00A0 C47704   LES    SI,04[BX]   引き数１の取り出し；ＥＳ・ＳＩにｓｄ１の番地を入れる
00A3 26       ES:                   (次の命令はエクストラセグメントベースを使う)
00A4 8B0C     MOV    CX,[SI]     ＣＸに引き数１の長さとリロケーションコードを入れる
00A6 D0E9     SHR    CL,1           引き数１の長さＣＬを２で割り余りをＣフラグに入れる
00A8 722F     JB     00D9           余りがあれば（Ｃフラグが１ならば）戻る
00AA 38C8     CMP    AL,CL          引き数１の長さの半分と引き数２の長さＡＬを比べる
00AC 752B     JNE    00D9           同じでないなら戻る
00AE 8CC2     MOV    DX,ES          ＤＸにｓｄ１のセグメントベースを取り出す
00B0 80FD00   CMP    CH,00       引き数１の文字列のセグメントベースを求める
00B3 7405     JE     00BA           ＣＨがゼロならＤＸの値をセグメントベースとする
00B5 30ED     XOR    CH,CH          ゼロでなければＣＨをクリアーする
00B7 BA6000   MOV    DX,0060        ＤＸにセグメントベースとして６０Ｈを設定する
00BA 26       ES:                   (次の命令はエクストラセグメントベースを使う)
00BB 8B7402   MOV    SI,02[SI]   ＳＩに引き数１の文字列の先頭番地のオフセットを入れる
00BE 8EDA     MOV    DS,DX       ＤＳに引き数１の文字列のセグメントベースを移す
00C0 07       POP    ES          ＥＳに引き数２の文字列のセグメントベースを戻す
00C1 5F       POP    DI          ＤＩに引き数２の文字列の先頭番地のオフセットを戻す
00C2 FC       CLD                ディレクションフラグを増加に設定する
00C3 AC       LODSB              ＡＬに引き数１の１文字を取り出す（ＳＩは１増える）
00C4 E81500   CALL   00DC        ＡＬの１６進数の数字を内部形に変換し結果をＡＬに入れる
00C7 51       PUSH   CX             （ＣＸを一時保存する）
00C8 B104     MOV    CL,04       ＣＬにシフトする桁数４をセットする
00CA D2E0     SAL    AL,CL       ＡＬを４ビット左にシフトする
00CC 59       POP    CX             （ＣＸをもとに戻す）
00CD 88C4     MOV    AH,AL          ＡＨに４ビットシフトした結果を移す
00CF AC       LODSB              ＡＬに引き数１の１文字を取り出す（ＳＩは１増える）
00D0 E80900   CALL   00DC        ＡＬの１６進数の数字を内部形に変換し結果をＡＬに入れる
00D3 08E0     OR     AL,AH       前に求めた上４ビットの値ＡＨと合成する
00D5 AA       STOSB              合成した値ＡＬを引き数２にしまう（ＤＩは１増える）
00D6 E2EB     LOOP   00C3        引き数２の長さＣＸだけ、以上の処理を繰り返す
00D8 CF       IRET               呼び出しプログラムに戻る
00D9 58       POP    AX          スタックを使っていた場合の戻り処理をする
00DA 58       POP    AX             （スタックポインターをもとに戻す）
00DB CF       IRET               呼び出しプログラムに戻る
```

内部サブルーチン

機能　　　：ＡＬにある１６進数の数字を内部形に変換し結果をＡＬに入れる
　　　　　　ＡＬの内容が１６進数の数字でないとき結果は意味がない
入り口状態：ＡＬ－１６進数の数字（文字コード）
戻り状態　：ＡＬ－変換結果（内部形）
　　　　　　ＡＬ・フラグ以外は変化しない
使用スタック：なし

```
00DC 2C30     SUB    AL,30       文字「０」～「９」が内部形０～９になるようにする
00DE 3C0A     CMP    AL,0A       結果が１０未満かどうかを調べる
00E0 7208     JB     00EA        結果が１０未満なら戻る
00E2 2C07     SUB    AL,07       文字「Ａ」～「Ｆ」が内部形１０～１５になるようにする
00E4 3C10     CMP    AL,10       結果が１６未満かどうかを調べる
00E6 7202     JB     00EA        結果が１６未満なら戻る
00E8 2C20     SUB    AL,20       文字「ａ」～「ｆ」が内部形１０～１５になるようにする
00EA C3       RET                呼び出し機械語プログラムに戻る
```

☆（９８）機械語　例題４　☆

１．機能

文字列を内部形２進数として受け取り、１６進数の数字（０〜９・Ａ〜Ｆ）に変換する。内部形１バイトが１６進数の数字２文字（２バイト）になる。

２．呼び出し形式

ＤＥＦ　ＳＥＧ＝＆Ｈ１Ｆ００
ＶＡＬＨＥＸ＝＆Ｈ１００
ＣＡＬＬ　ＶＡＬＨＥＸ（文字列変数１，文字列変数２）
文字列変数１（入力）　：内部形２進数として扱われる文字列
文字列変数２（出力）　：変換した１６進数の数字を入れる文字列（長さは文字列変数１の長さの２倍であること）

３．利用例

整数型・単精度実数型・倍精度実数型の数値の内部形式を表示する。

```
100 DEF SEG=&H1F00: VALHEX=&H100
110 FOR I=-3 TO 3
120   VI$=MKI$(I): HI$=SPACE$( 4): CALL VALHEX(VI$,HI$)
130   VS$=MKS$(I): HS$=SPACE$( 8): CALL VALHEX(VS$,HS$)
140   VD$=MKD$(I): HD$=SPACE$(16): CALL VALHEX(VD$,HD$)
150   PRINT I,HI$,HS$,HD$
160 NEXT I
run
-3              FDFF          0000C082        000000000000C082
-2              FEFF          00008082        0000000000008082
-1              FFFF          00008081        0000000000008081
 0              0000          00000000        0000000000000000
 1              0100          00000081        0000000000000081
 2              0200          00000082        0000000000000082
 3              0300          00004082        0000000000004082
```

４．制約事項

・引き数の数や型が上の呼び出し形式と違う場合、正しく動かない。
・文字列変数２の長さが文字列変数１の長さの２倍でない場合はなにもしない。

５．機械語プログラム　　入り口状態　：ＢＸ−　引き数テーブルのオフセット
ＤＳ−　引き数テーブルのセグメントベース
ＣＳ−　コードセグメントベース（値１Ｆ００Ｈ）
引き数テーブル−　引き数２、引き数１の順でそれぞれのストリングディスクリプター（ｓｄ２・ｓｄ１）のオフセット・セグメントベースが入っている

ＣＳ＝１Ｆ００

```
0100 C437    LES   SI,[BX]     引き数２の取り出し；ＥＳ・ＳＩにｓｄ２の番地を入れる
0102 26      ES:                 (次の命令はエクストラセグメントベースを使う)
0103 8B0C    MOV   CX,[SI]     ＣＸに引き数２の長さとリロケーションコードを取り出す
0105 80F900  CMP   CL,00          引き数２の長さＣＬがゼロかどうか調べる
0108 744C    JE    0156           引き数２の長さがゼロなら戻る
010A D0E9    SHR   CL,1           引き数２の長さＣＬを２で割り余りをＣフラグに入れる
010C 7248    JB    0156           余りがあれば（Ｃフラグが１ならば）戻る
010E 88C8    MOV   AL,CL          ＡＬに引き数２の長さを２で割った値ＣＬを保存する
0110 8CC2    MOV   DX,ES          ＤＸにｓｄ２のセグメントベースを入れる
0112 80FD00  CMP   CH,00       引き数２の文字列のセグメントベースを求める
0115 7405    JE    011C           ＣＨがゼロならばＤＸの値をセグメントベースとする
0117 30ED    XOR   CH,CH          ゼロでなければＣＨをクリアーする
0119 BA6000  MOV   DX,0060        ＤＸにセグメントベースとして６０Ｈを設定する
011C 26      ES:                 (次の命令はエクストラセグメントベースを使う)
011D 8B7C02  MOV   DI,02[SI]   ＤＩに引き数２の文字列の先頭番地のオフセットを入れる
0120 8EC2    MOV   ES,DX       ＥＳに引き数２の文字列のセグメントベースを入れる
0122 57      PUSH  DI             引き数２の文字列の先頭番地のオフセットＤＩをしまう
0123 06      PUSH  ES             引き数２の文字列のセグメントベースＥＳをしまう
0124 C47704  LES   SI,04[BX]   引き数１の取り出し；ＥＳ・ＳＩにｓｄ１の番地を入れる
0127 26      ES:                 (次の命令はエクストラセグメントベースを使う)
```

```
0128 8B0C     MOV    CX,[SI]      ＣＸに引き数１の長さとリロケーションコードを取り出す
012A 38C8     CMP    AL,CL          引き数１の長さと引き数２の長さの半分ＣＬを比べる
012C 7529     JNE    0157           同じでないなら戻る
012E 8CC2     MOV    DX,ES          ＤＸにｓｄ１のセグメントベースを入れる
0130 80FD00   CMP    CH,00        引き数１の文字列のセグメントベースを求める
0133 7405     JE     013A           ＣＨがゼロならばＤＸの値をセグメントベースとする
0135 30ED     XOR    CH,CH          ゼロでなければＣＨをクリアーする
0137 BA6000   MOV    DX,0060        ＤＸにセグメントベースとして６０Ｈをセットする
013A 26       ES:                  (次の命令はエクストラセグメントベースを使う)
013B 8B7402   MOV    SI,02[SI]    ＳＩに引き数１の文字列の先頭番地のオフセットを入れる
013E 8EDA     MOV    DS,DX        ＤＳに引き数１のセグメントベースを移す
0140 07       POP    ES           ＥＳに引き数２の文字列のセグメントベースを戻す
0141 5F       POP    DI           ＤＩに引き数２の文字列の先頭番地のオフセットを戻す
0142 FC       CLD                 ディレクションフラグを増加に設定する
0143 8A04     MOV    AL,[SI]      ＡＬに引き数１から１文字取り出す
0145 51       PUSH   CX             (ＣＸを一時保存する)
0146 B104     MOV    CL,04        ＣＬにシフトする桁数４をセットする
0148 D2E8     SHR    AL,CL        ＡＬを４ビット右にシフトする
014A 59       POP    CX             (ＣＸをもとに戻す)
014B E80C00   CALL   015A         ＡＬの下４ビットを１６進数の数字１文字に変換する
014E AA       STOSB               変換した値ＡＬを引き数２にしまう（ＤＩは１増える）
014F AC       LODSB               ＡＬに同じ文字を取り出す（ＳＩは次の文字に進む）
0150 E80700   CALL   015A         ＡＬの下４ビットを１６進数の数字１文字に変換する
0153 AA       STOSB               変換した値ＡＬを引き数２にしまう（ＤＩは１増える）
0154 E2ED     LOOP   0143         引き数１の長さだけ、以上の処理を繰り返す
0156 CF       IRET                呼び出しプログラムに戻る
0157 58       POP    AX           スタックを使っていた場合の戻り処理をする
0158 58       POP    AX             (スタックポインターをもとに戻す)
0159 CF       IRET                呼び出しプログラムに戻る
```

内部サブルーチン

機能　　　　　：ＡＬの下位４ビットにある２進数を１６進数の数字に変換しＡＬに返す
入り口状態　：ＡＬ－変換する値（下位４ビットが有効・上位４ビットは無視）
戻り状態　　：ＡＬ－変換した結果の１６進数の数字
　　　　　　　　ＡＬ・フラグ以外は変化しない
使用スタック：なし

```
015A 240F     AND    AL,0F        ＡＬの下位４ビットを取り出す
015C 0430     ADD    AL,30        内部形０～９が文字「０」～「９」になるようにする
015E 3C3A     CMP    AL,3A        「９」以下（文字コード３ＡＨ未満）かどうか調べる
0160 7202     JB     0164         「９」以下なら戻る
0162 0407     ADD    AL,07        内部形１０～１５が文字「Ａ」～「Ｆ」になるようにする
0164 C3       RET                 呼び出し機械語プログラムに戻る
```

☆（９８）機械語　例題５　☆

１．機能

セグメントベースならびにオフセットで指定した番地からメモリーの内容を文字列に取り出す。取り出す長さは文字列の長さとする。

２．呼び出し形式

```
DEF SEG=&H1F00
PEEKL=&H180
CALL PEEKL（整数変数１，整数変数２，文字列変数）
```

整数変数１（入力）　：メモリーの内容を取り出す開始番地のセグメントベース
整数変数２（入力）　：メモリーの内容を取り出す開始番地のオフセット
文字列変数（出力）　：取り出したメモリーの内容を入れる文字列（長さが取り出す長さ）

3．利用例

指定したメモリーの内容を１６進数と文字形式でプリンターに印字する。
例題２（ＵＳＲ２）・例題４（ＶＡＬＨＥＸ）も利用する。

```
100 DEF SEG=&H1F00: DEF USR2=&H40: VALHEX=&H100: PEEKL=&H180
110 INPUT "ｾｸﾞﾒﾝﾄ";SS%
120 INPUT "ｶｲｼ ﾊﾞﾝﾁ,ｼｭｳﾘｮｳ ﾊﾞﾝﾁ";S,E
130 FOR I%=S TO E STEP &H10
140  IF E-I%+1>&H10 THEN L=&H10 ELSE L=E-I%+1
150  V$=SPACE$(L): CALL PEEKL(SS%,I%,V$)
160  H$=SPACE$(L*2): CALL VALHEX(V$,H$)
170  LPRINT HEX$(I%);TAB(6);USR2(V$);TAB(24);H$
180 NEXT I%
run
ｾｸﾞﾒﾝﾄ? &he800
ｶｲｼ ﾊﾞﾝﾁ,ｼｭｳﾘｮｳ ﾊﾞﾝﾁ? &h690,&h6d9
690    ··KCNV$L··SAVE__   00054B434E56244C0004534156458180
6A0    ·LOAD▄_·EEP▄_·ON   044C4F41448280034545508380064F4E
6B0    SOLE▄_·OPY▄_·LOS   534F4C458480034F50598580044C4F53
6C0    E▄_·ONT█_·LEARI _  458680034F4E548780044C4541528880
6D0    ·SRLIN·_·I         0553524C494E14800349
```

4．制約事項

引き数の数や型が上の呼び出し形式と違う場合、正しく動かない。

5．機械語プログラム　　入り口状態　：ＢＸ－　引き数テーブルのオフセット
ＤＸ－　引き数テーブルのセグメントベース
ＣＳ－　コードセグメントベース（値１Ｆ００Ｈ）
引き数テーブル－　引き数３のストリングディスクリプター（ｓｄ）のオフセット・セグメントベース、引き数２・引き数１のオフセット・セグメントベースが順に入っている

```
ＣＳ＝１Ｆ００
0180 C47704   LES    SI,04[BX]   引き数２の取り出し；ＥＳ・ＳＩに引き数２の番地を入れる
0183 26       ES:                  (次の命令はエクストラセグメントベースを使う)
0184 8B04     MOV    AX,[SI]       ＡＸに引き数２の値を取り出す
0186 C47708   LES    SI,08[BX]   引き数１の取り出し；ＥＳ・ＳＩに引き数１の番地を入れる
0189 26       ES:                  (次の命令はエクストラセグメントベースを使う)
018A 8B14     MOV    DX,[SI]       ＤＸに引き数１の値を取り出す
018C C437     LES    SI,[BX]     引き数３の取り出し；ＥＳ・ＳＩにｓｄの番地を入れる
018E 26       ES:                  (次の命令はエクストラセグメントベースを使う)
018F 8B0C     MOV    CX,[SI]       ＣＸに引き数３の長さとリロケーションコードを入れる
0191 80F900   CMP    CL,00         引き数３の長さがゼロかどうか調べる
0194 7418     JE     01AE          引き数３の長さがゼロならば戻る
0196 8CC3     MOV    BX,ES         ＢＸにｓｄのセグメントベースを入れる
0198 80FD00   CMP    CH,00       引き数３の文字列のセグメントベースを求める
019B 7405     JE     01A2          ＣＨがゼロならばＢＸの値をセグメントベースとする
019D 30ED     XOR    CH,CH         ゼロでなければＣＨをクリアーする
019F BB6000   MOV    BX,0060       ＢＸにセグメントベースとして６０Ｈを設定する
01A2 26       ES:                  (次の命令はエクストラセグメントベースを使う)
01A3 8B7C02   MOV    DI,02[SI]   ＤＩに引き数３の文字列の先頭番地のオフセットを入れる
01A6 8EC3     MOV    ES,BX       ＥＳに引き数３の文字列のセグメントベースを入れる
01A8 8EDA     MOV    DS,DX       ＤＳに対象番地のセグメントベースを入れる
01AA 89C6     MOV    SI,AX       ＳＩに対象番地のオフセットを入れる
01AC F3       REP                引き数１・２で指定された番地から引き数３の文字列へ文字
01AD A4       MOVSB                列の長さＣＸだけメモリーの内容を移す
01AE CF       IRET               呼び出しプログラムに戻る
```

☆（９８）機械語　例題６　☆

１．機能

文字列の内容を指定した番地からメモリーに書き込む。書き込む長さは文字列の長さとする。

２．呼び出し形式

ＤＥＦ　ＳＥＧ＝＆Ｈ１Ｆ００
ＰＯＫＥＬ＝＆Ｈ１Ｃ０
ＣＡＬＬ　ＰＯＫＥＬ（文字列変数，整数変数１，整数変数２）
　　文字列変数（入力）　：メモリーに書き込む内容（長さが書き込む長さ）
　　整数変数１（入力）　：メモリーに書き込む開始番地のセグメントベース
　　整数変数２（入力）　：メモリーに書き込む開始番地のオフセット

３．利用例

機械語例題１（ＵＳＲ１）の機械語プログラムをメモリーに書き込む。例題３（ＨＥＸＶＡＬ）も利用する。

```
100 DEF SEG=&H1F00: HEXVAL=&H80: POKEL=&H1C0
110 H$="3c0275128b07b9100031d2d1e0730142e2f98917b002cf"
120 V$=SPACE$(LEN(H$)/2): CALL HEXVAL(H$,V$)
130 S%=&H1F00: A%=&H0: CALL POKEL(V$,S%,A%)
```

４．制約事項

・引き数の数や型が上の呼び出し形式と違う場合、正しく動かない。
・文字列変数が空文字列（ヌルストリング）の場合なにもしない。

５．機械語プログラム　　入り口状態　：ＢＸ－　引き数テーブルのオフセット
　　ＤＳ－　引き数テーブルのセグメントベース
　　ＣＳ－　コードセグメントベース（値１Ｆ００Ｈ）
　　引き数テーブル－　引き数３・２のオフセット・セグメントベース、引き数１のストリングディスクリプター（ｓｄ）のオフセット・セグメントベースが順に入っている

```
CS=1F00
01C0 C47704   LES    SI,04[BX]   引き数２の取り出し；ＥＳ・ＳＩに引き数２の番地を入れる
01C3 26       ES:                 (次の命令はエクストラセグメントベースを使う)
01C4 8B14     MOV    DX,[SI]       ＤＸに引き数２の値を取り出す
01C6 C437     LES    SI,[BX]     引き数３の取り出し；ＥＳ・ＳＩに引き数１の番地を入れる
01C8 26       ES:                 (次の命令はエクストラセグメントベースを使う)
01C9 8B04     MOV    AX,[SI]       ＡＸに引き数１の値を取り出す
01CB C47708   LES    SI,08[BX]   引き数１の取り出し；ＥＳ・ＳＩにｓｄの番地を入れる
01CE 26       ES:                 (次の命令はエクストラセグメントベースを使う)
01CF 8B0C     MOV    CX,[SI]       ＣＸに引き数１の長さとリロケーションコードを入れる
01D1 80F900   CMP    CL,00         引き数１の長さＣＬがゼロかどうか調べる
01D4 7418     JE     01EE          引き数１の長さＣＬがゼロならば戻る
01D6 8CC3     MOV    BX,ES         ＢＸにｓｄのセグメントベースを入れる
01D8 80FD00   CMP    CH,00       引き数１の文字列のセグメントベースを求める
01DB 7405     JE     01E2          ＣＨがゼロならばＢＸの値をセグメントベースとする
01DD 30ED     XOR    CH,CH         ゼロでなければＣＨをクリアーする
01DF BB6000   MOV    BX,0060       ＢＸにセグメントベースとして６０Ｈを設定する
01E2 26       ES:                 (次の命令はエクストラセグメントベースを使う)
01E3 8B7402   MOV    SI,02[SI]   ＳＩに引き数１の文字列のオフセットを入れる
01E6 8EDB     MOV    DS,BX       ＤＳに引き数１の文字列のセグメントベースを入れる
01E8 8EC2     MOV    ES,DX       ＥＳに対象番地のセグメントベースを入れる
01EA 89C7     MOV    DI,AX       ＤＩに対象番地のオフセットを入れる
01EC F3       REP                引き数１の文字列から引き数２・３で指定された番地へ文字
01ED A4       MOVSB              列の長さＣＸだけ文字列の内容を移す
01EE CF       IRET               呼び出しプログラムに戻る
```

☆（９８）機械語　例題７　☆

1．機能

文字列のなかにある特定の文字を別の文字に置き換える。引き数１は操作の対象となる文字列で処理結果も引き数１に戻る。引き数２と引き数３の文字列は先頭から１文字ずつ対応し、引き数２に置き換えの対象となる文字を指定する。引き数３は置き換える文字を指定する。

2．呼び出し形式

ＤＥＦ　ＳＥＧ＝＆Ｈ１Ｆ００
ＲＥＰＬＡＣＥ＝＆Ｈ２００
ＣＡＬＬ　ＲＥＰＬＡＣＥ（文字列変数１，文字列変数２，文字列変数３）

文字列変数１（入力／出力）　：処理の対象となる文字列
文字列変数２（入力）　：置き換えの対象となる文字（２文字以上指定した場合文字列変数３の先頭から１文字ずつ対応する）
文字列変数３（入力）　：置き換える文字（長さは文字列変数２と同じであること）

3．利用例

キーボードから入力した文字列の中のアルファベットを大文字に変換して画面に表示する。

```
100 DEF SEG=&H1F00: REPLACE=&H200
110 LC$="abcdefghijklmnopqrstuvwxyz"
120 UC$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
130 LINE INPUT "ﾓｼﾞﾚﾂ ﾊ? ";R$
140 CALL REPLACE(R$,LC$,UC$): PRINT R$
run
ﾓｼﾞﾚﾂ ﾊ? Asahi Shimbun Denshi-keisan-shitsu
ASAHI SHIMBUN DENSHI-KEISAN-SHITSU
```

4．制約事項

・引き数の数や型が上の呼び出し形式と違う場合、正しく動かない。
・文字列変数２と文字列変数３の長さが異なる場合はなにもしない。
・引き数の中に空文字列（ヌルストリング）がある場合はなにもしない。
・文字列変数２の中に同じ文字がある場合は最初の文字に対応する文字に置き換える。

5．機械語プログラム　　入り口状態　：ＢＸ－　引き数テーブルのオフセット
ＤＳ－　引き数テーブルのセグメントベース
ＣＳ－　コードセグメントベース（値１Ｆ００Ｈ）
引き数テーブル－　引き数３、引き数２、引き数１の順でそれぞれのストリングディスクリプター（ｓｄ３・ｓｄ２・ｓｄ１）のオフセット・セグメントベースが入っている

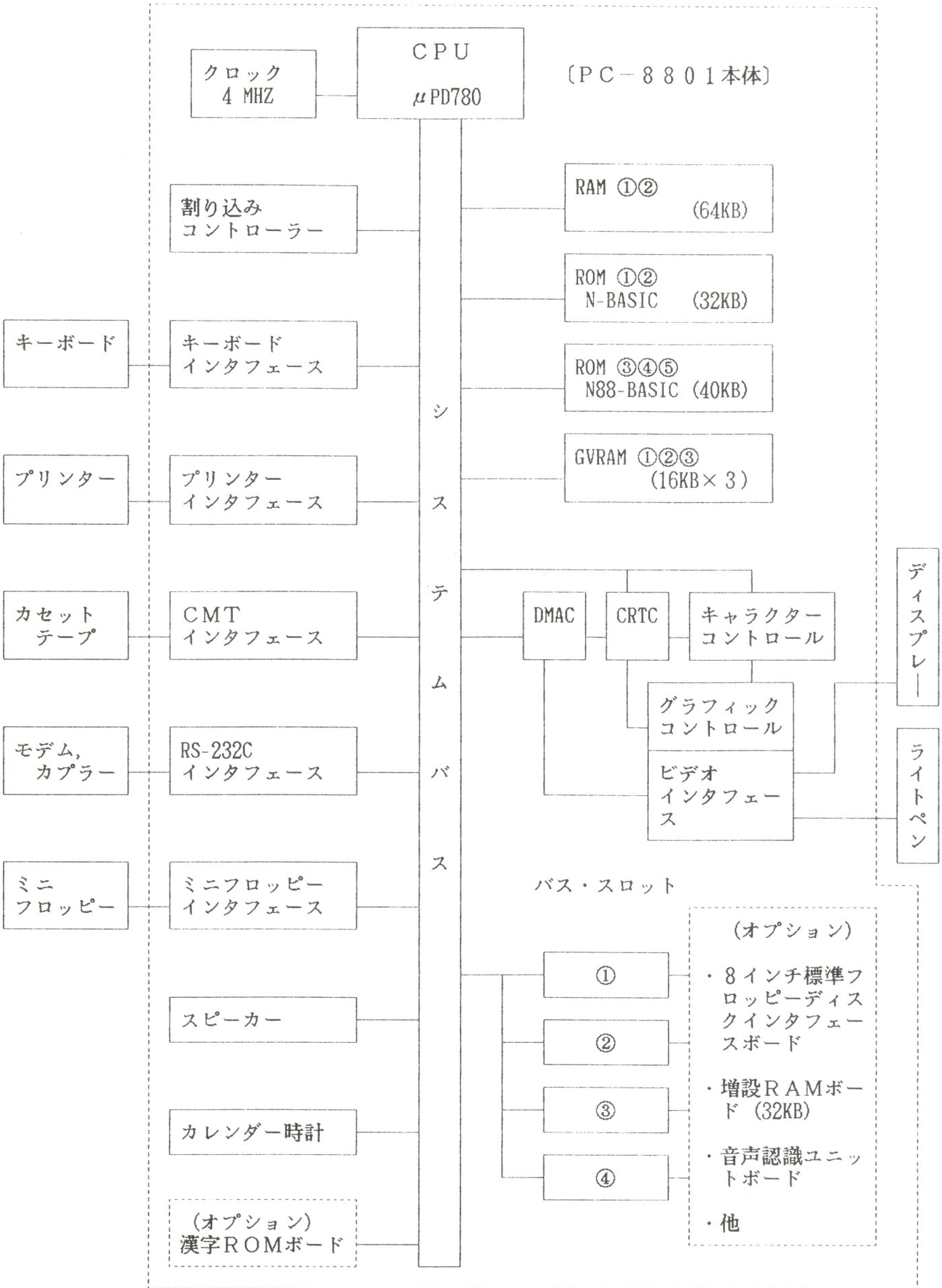
```
CS=1F00
0200 1E         PUSH  DS          引き数テーブルのセグメントベースをスタックに保存する
0201 0E         PUSH  CS          ＥＳに機械語エリアのセグメントベースＣＳを移す
0202 07         POP   ES
0203 C57708     LDS   SI,08[BX]   引き数１の取り出し；ＤＳ・ＳＩにｓｄ１の番地を入れる
0206 8B0C       MOV   CX,[SI]       ＣＸに引き数１の長さとリロケーションコードを入れる
0208 80F900     CMP   CL,00         引き数１の文字列１の長さＣＬがゼロかどうか調べる
020B 7428       JE    0235          引き数１の文字列の長さがゼロならば戻る
020D 8CDA       MOV   DX,DS         ＤＸにｓｄ１のセグメントベースを移す
020F 80FD00     CMP   CH,00       引き数１の文字列のセグメントベースを求める
0212 7405       JE    0219          ＣＨがゼロならばＤＸの値をセグメントベースとする
0214 30ED       XOR   CH,CH         ゼロでなければＣＨをクリアーする
0216 BA6000     MOV   DX,0060       ＤＸにセグメントベースとして６０Ｈを設定する
0219 8B7402     MOV   SI,02[SI]     ＳＩに引き数１の文字列の先頭番地のオフセットを移す
021C 26         ES:                （次の命令はエクストラセグメントベースを使う）
021D 880E9F02   MOV   [029F],CL     引き数１の文字列の長さＣＬを保存する
0221 26         ES:                （次の命令はエクストラセグメントベースを使う）
0222 8916A002   MOV   [02A0],DX     引き数１の文字列のセグメントベースＤＸを保存する
0226 26         ES:                （次の命令はエクストラセグメントベースを使う）
0227 8936A202   MOV   [02A2],SI     引き数１の文字列の先頭番地のオフセットＳＩを保存
022B 1F         POP   DS          ＤＳに引き数テーブルのセグメントベースを戻す
022C 1E         PUSH  DS          引き数テーブルのセグメントベースＤＳを保存しなおす
022D C57704     LDS   SI,04[BX]   引き数２の取り出し；ＤＳ・ＳＩにｓｄ２の番地を入れる
```

```
0230 8B0C      MOV   CX,[SI]      ＣＸに引き数２の長さとリロケーションコードを入れる
0232 80F900    CMP   CL,00        引き数２の文字列の長さＣＬがゼロかどうか調べる
0235 7466      JE    029D         引き数２の文字列の長さＣＬがゼロならば戻る
0237 88C8      MOV   AL,CL        ＡＬに引き数２の文字列の長さＣＬを移す
0239 8CDA      MOV   DX,DS        ＤＸにｓｄ２のセグメントベースを移す
023B 80FD00    CMP   CH,00      引き数２の文字列のセグメントベースを求める
023E 7405      JE    0245         ＣＨがゼロならばＤＸの値をセグメントベースとする
0240 30ED      XOR   CH,CH        ゼロでなければＣＨをクリアーする
0242 BA6000    MOV   DX,0060      ＤＸにセグメントベースとして６０Ｈを設定する
0245 8B7402    MOV   SI,02[SI]    ＳＩに引き数２の文字列の先頭番地のオフセットを移す
0248 26        ES:               (次の命令はエクストラセグメントベースを使う)
0249 890EA402  MOV   [02A4],CX    引き数２の文字列の長さＣＸを保存する
024D 8EDA      MOV   DS,DX        ＤＳに引き数２の文字列のセグメントベースを移す
024F FC        CLD                ディレクションフラグを増加に設定する
0250 BFA602    MOV   DI,02A6      ＤＩに機械語エリアの作業場所のオフセットを入れる
0253 F3        REP                引き数２の文字列の内容を機械語エリアの作業場所にコ
0254 A4        MOVSB              ピーする
0255 1F        POP   DS         ＤＳに引き数テーブルのセグメントベースを戻す
0256 C537      LDS   SI,[BX]    引き数３の取り出し；ＤＳ・ＳＩにｓｄ３の番地を入れる
0258 8B0C      MOV   CX,[SI]      ＣＸに引き数３の長さとリロケーションコードを入れる
025A 38C1      CMP   CL,AL        引き数３の長さＣＬと引き数２の長さＡＬを比べる
025C 753E      JNE   029C         引き数３と２の長さが同じでなければ戻る
025E 8CDA      MOV   DX,DS        ＤＸにｓｄ３のセグメントベースを移す
0260 80FD00    CMP   CH,00      引き数３の文字列のセグメントベースを求める
0263 7405      JE    026A         ＣＨがゼロならばＤＸの値をセグメントベースとする
0265 30ED      XOR   CH,CH        ゼロでなければＣＨをクリアーする
0267 BA6000    MOV   DX,0060      ＤＸにセグメントベースとして６０Ｈを設定する
026A 8B7402    MOV   SI,02[SI]    ＳＩに引き数３の文字列の先頭番地のオフセットを移す
026D 8EDA      MOV   DS,DX        ＤＳに引き数３の文字列のセグメントベースを移す
026F BFA603    MOV   DI,03A6      ＤＩに機械語エリアの作業場所のオフセットを入れる
0272 F3        REP                引き数３の文字列の内容を機械語エリアの作業場所にコ
0273 A4        MOVSB              ピーする
0274 26        ES:               (次の命令はエクストラセグメントベースを使う)
0275 8E1EA002  MOV   DS,[02A0]  ＤＳに引き数１の文字列のセグメントベースを移す
0279 26        ES:               (次の命令はエクストラセグメントベースを使う)
027A 8B36A202  MOV   SI,[02A2]  ＳＩに引き数１の文字列の先頭番地のオフセットを移す
027E 26        ES:               (次の命令はエクストラセグメントベースを使う)
027F 8B0EA402  MOV   CX,[02A4]  ＣＸに引き数２の文字列の長さを取り出す
0283 8A04      MOV   AL,[SI]    ＡＬに引き数１の文字列の１文字を取り出す
0285 BFA602    MOV   DI,02A6    ＤＩに機械語エリアに移した引き数２のオフセットを入れる
0288 F2        REPNE            引き数１の文字列の１文字ＡＬと同じ文字が引き数２の文字
0289 AE        SCASB            列の中にあるかどうか調べる
028A 7508      JNE   0294       同じ文字がない場合、引き数１の次の文字の取り出しへ行く
028C 4F        DEC   DI         ＤＩを見つけた文字のオフセットにする
028D 26        ES:               (次の命令はエクストラセグメントベースを使う)
028E 8A850001  MOV   AL,0100[DI] 引き数１の１文字を引き数３の対応する文字（100Hは機械語
0292 8804      MOV   [SI],AL      エリアに移した引き数２と３の番地の差）に置き換える
0294 46        INC   SI         ＳＩを引き数１の文字列の次の文字のオフセットにする
0295 26        ES:               (次の命令はエクストラセグメントベースを使う)
0296 FE0E9F02  DEC   BYTE [029F] 引き数１の残り文字数から１を引く
029A 75E2      JNE   027E       処理していない文字が残っていれば繰り返す
029C CF        IRET             呼び出しプログラムに戻る
029D 58        POP   AX         スタックを戻す
029E CF        IRET             呼び出しプログラムに戻る
```

作業領域

２９Ｆ	１バイト	引き数１の未処理文字数
２Ａ０	２バイト	引き数１のセグメントベース
２Ａ２	２バイト	引き数１のオフセット
２Ａ４	２バイト	引き数２の文字列の長さ
２Ａ６～３Ａ４	２５５バイト	引き数２の文字列の内容の写し
３Ａ６～４Ａ４	２５５バイト	引き数３の文字列の内容の写し

1. 8801 ブロック構成図



2．8801　I/Oアドレスマップ

アドレス	接続機器　［使用ＬＳＩ］
００Ｈ ～０ＢＨ	キーボード　▷「キーボードマップ」参照
１０Ｈ	プリンターインタフェース［ＴＴＬレベル］／カレンダー時計［μPD1990AC相当］ ▷システムポート40H でプリンター／カレンダー時計の選択をする　▷プリンターへのデータの出力　▷カレンダー時計の動作制御と時刻のセット　▷ＢＡＳＩＣのＤＡＴＥ＄・ＴＩＭＥ＄などで参照する
２０Ｈ ２１Ｈ	ＣＭＴインタフェース／ＲＳ－２３２Ｃインタフェース　［μPD8251C 相当］ ▷システムポート40H でＣＭＴ／ＲＳ－２３２Ｃの選択をする　▷インタフェースの動作制御とデータの入出力　▷ＣＭＴのボーレート、モーターのＯＮ／ＯＦＦなどはシステムポート30H で指定する　▷ＢＡＳＩＣのＣＬＯＡＤ・ＣＳＡＶＥなどで参照する ▷ＢＡＳＩＣのＣＯＭ：で参照する
３０Ｈ ３１Ｈ	システムポート ▷本体裏のディップスイッチの状態の入力　▷ディスプレー・ＣＭＴ・ＲＳ－２３２Ｃインタフェースへの各種制御信号の出力　▷メモリー構成の指定
４０Ｈ	システムポート ▷プリンター・ＲＳ－２３２Ｃ・PC-8031 ミニフロッピーディスクユニット・カレンダー時計インタフェースからの各種制御信号の入力　▷プリンター・ブザー・カレンダー時計への各種制御信号の出力
５０Ｈ ¦ ５ＦＨ	ＣＲＴコントローラー　［μPD3301D-2 相当］ ▷ボーダーカラー・バックグラウンドカラー・カラーパレットの指定　▷メモリーアドレス C000H～FFFFH に対応するグラフィックＶＲＡＭの切り替え
６０Ｈ ¦ ６８Ｈ	ＤＭＡコントローラー　［μPD8257C-5 相当］ ▷ＣＲＴインタフェースとＶＲＡＭ間のデータ転送をする　▷８インチフロッピーディスクとメモリー間のデータを転送する　▷ＣＰＵを通さずに直接メモリーと周辺機器の間でデータ転送することをＤＭＡという
７０Ｈ ７１Ｈ ７８Ｈ	メモリーコントロール ▷メモリーアドレス 6000H～7FFFH に対応する拡張ＲＯＭの選択　▷メモリーアドレス 8000H～83FFH に対応するＢＡＳＩＣテキストＲＡＭエリアの選択
Ｅ２Ｈ	メモリーコントロール ▷メモリーアドレス 8000H～83FFH に対応するＲＯＭ／ＢＡＳＩＣテキストＲＡＭの指定
Ｅ４Ｈ	割り込みコントローラー　［μPB8214C 相当］ ▷ＣＭＴ／ＲＳ－２３２Ｃデータ受信・ＣＲＴ垂直帰線サイクル・インタバルタイマー・８インチフロッピーディスクからの割り込みを管理する　▷Ｎ－ＢＡＳＩＣ使用時は割り込みを禁止している
Ｅ６Ｈ	割り込み制御 ▷ＣＭＴ／ＲＳ－２３２Ｃデータ受信・ＣＲＴ垂直帰線サイクル・インタバルタイマーによる割り込みの許可／禁止を設定する
Ｅ８Ｈ ～ＥＢＨ	文字パターンＲＯＭ　［μPD23C128C 相当］ ▷オプションボード PC-8801-01
ＦＣＨ ¦ ＦＦＨ	５インチフロッピーディスクインタフェース　［μPD8255AC-5相当］ ▷５インチフロッピーディスクインタフェースの動作制御とデータの入出力　▷５インチフロッピーディスクユニット PC-8031,PC-8031-2W などを接続する

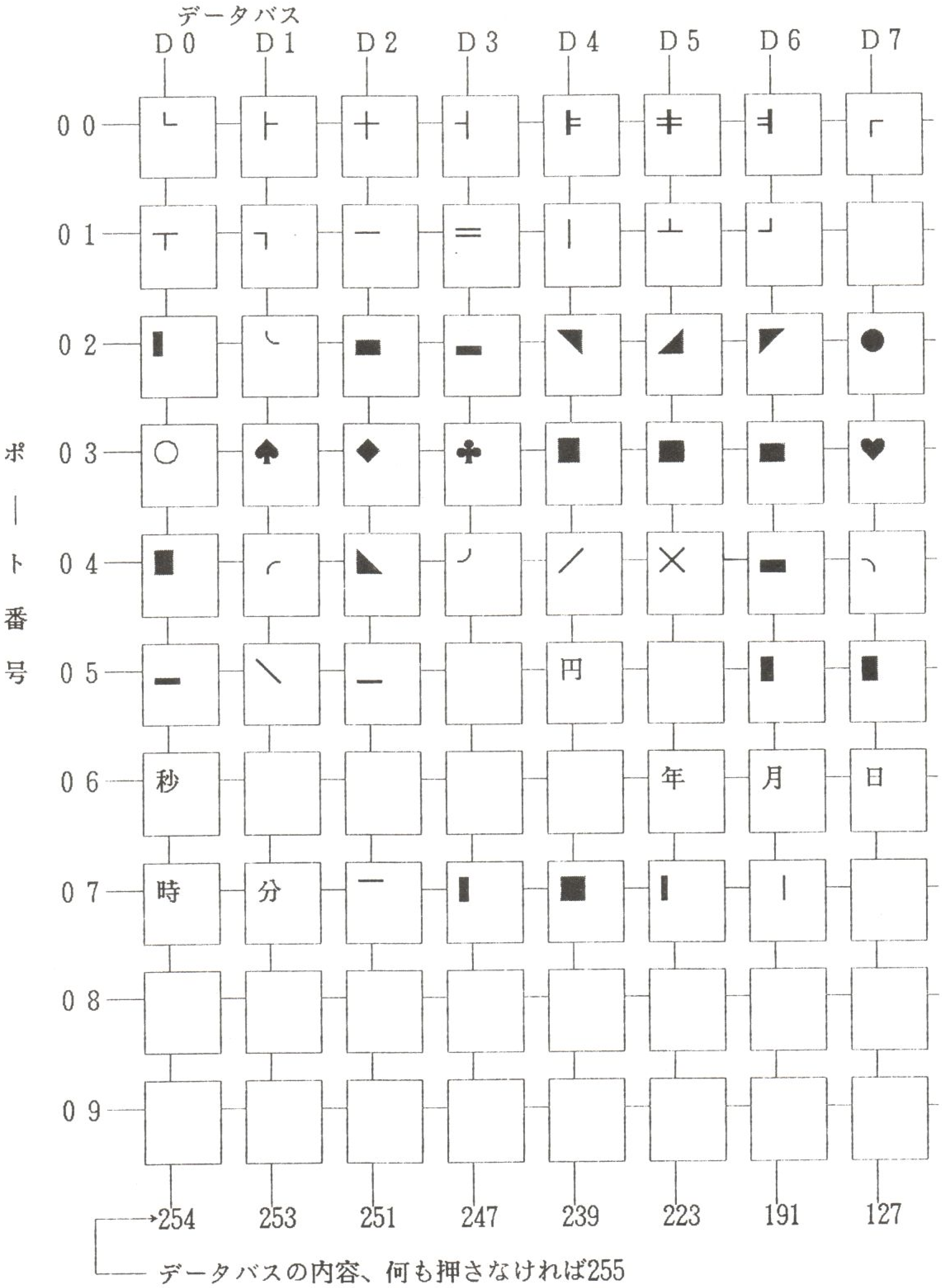
3. 8801 キーボードマップ

データバス

ポート番号	D0	D1	D2	D3	D4	D5	D6	D7
00	0	1	2	3	4	5	6	7
01	8	9	*	+	=	,	.	↵
02	@ ゛	A チ	B コ	C ソ	D シ	E イ	F ハ	G キ
03	H ク	I ニ	J マ	K ノ	L リ	M モ	N ミ	O ラ
04	P セ	Q タ	R ス	S ト	T カ	U ナ	V ヒ	W テ
05	X サ	Y ン	Z ッ ツ	「 [゜	¥ ー	」] ム	^ ヘ	= − ホ
06	ヲ 0 ワ	! 1 ヌ	" 2 フ	# ァ 3 ア	$ ゥ 4 ウ	% ェ 5 エ	& ォ 6 オ	' ャ 7 ヤ
07	(ュ 8 ユ	) ョ 9 ヨ	* : ケ	+ ; レ	< 、 , ネ	> 。 . ル	? ・ / メ	_ ロ
08	HOME CLR	↑	→	INS DEL	GRPH	カナ	SHIFT	CTRL
09	STOP	f.1	f.2	f.3	f.4	f.5	SPACE	ESC
0A	TAB	↓	←	HELP	COPY	−	/	CAPS LOCK
0B	ROLL UP	ROLL DOWN						
	254	253	251	247	239	223	191	127

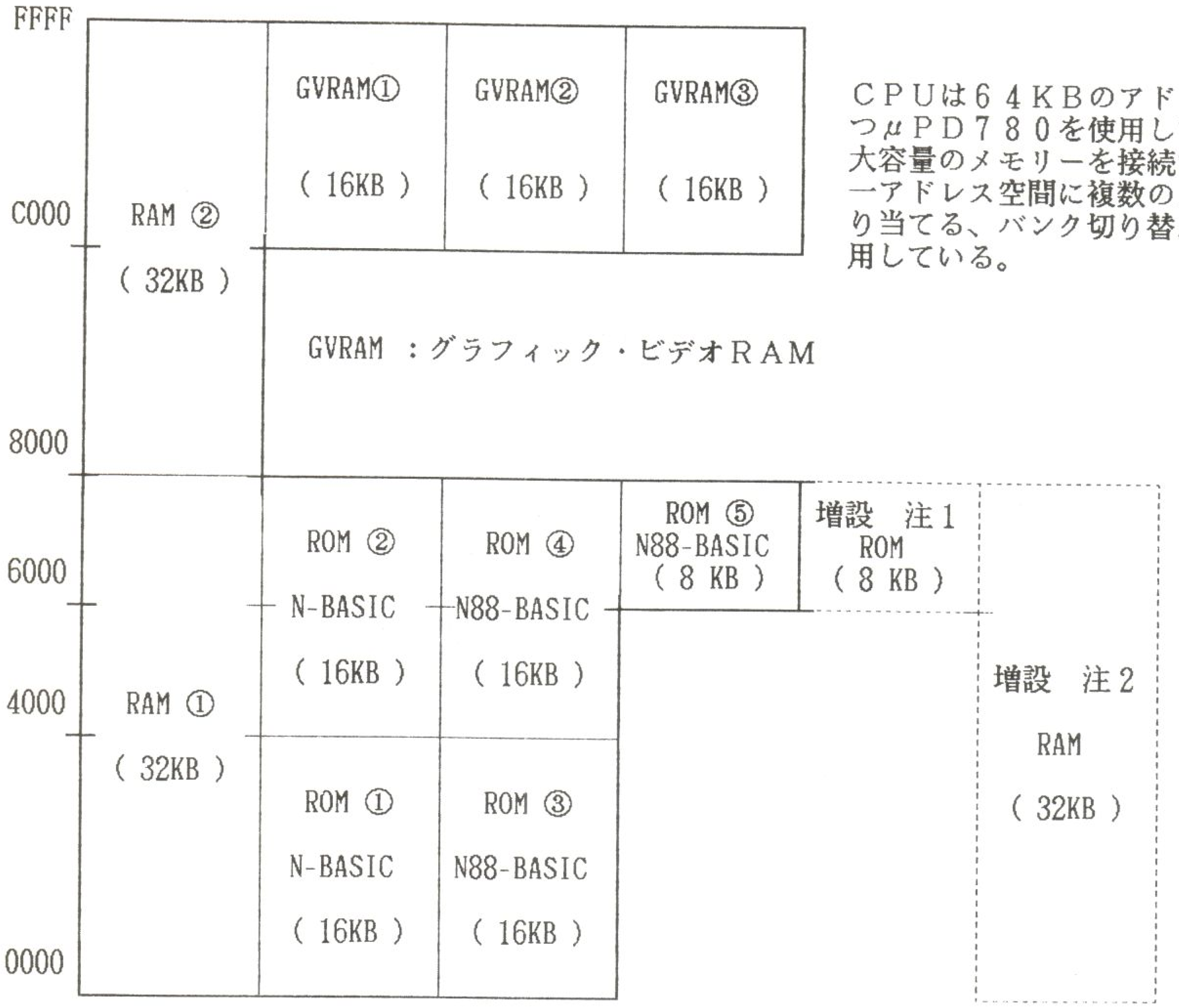
データバスの内容、何も押さなければ255

（グラフィックキャラクターモード）



4．8801　メモリーアドレスマップ

☆アドレス空間☆

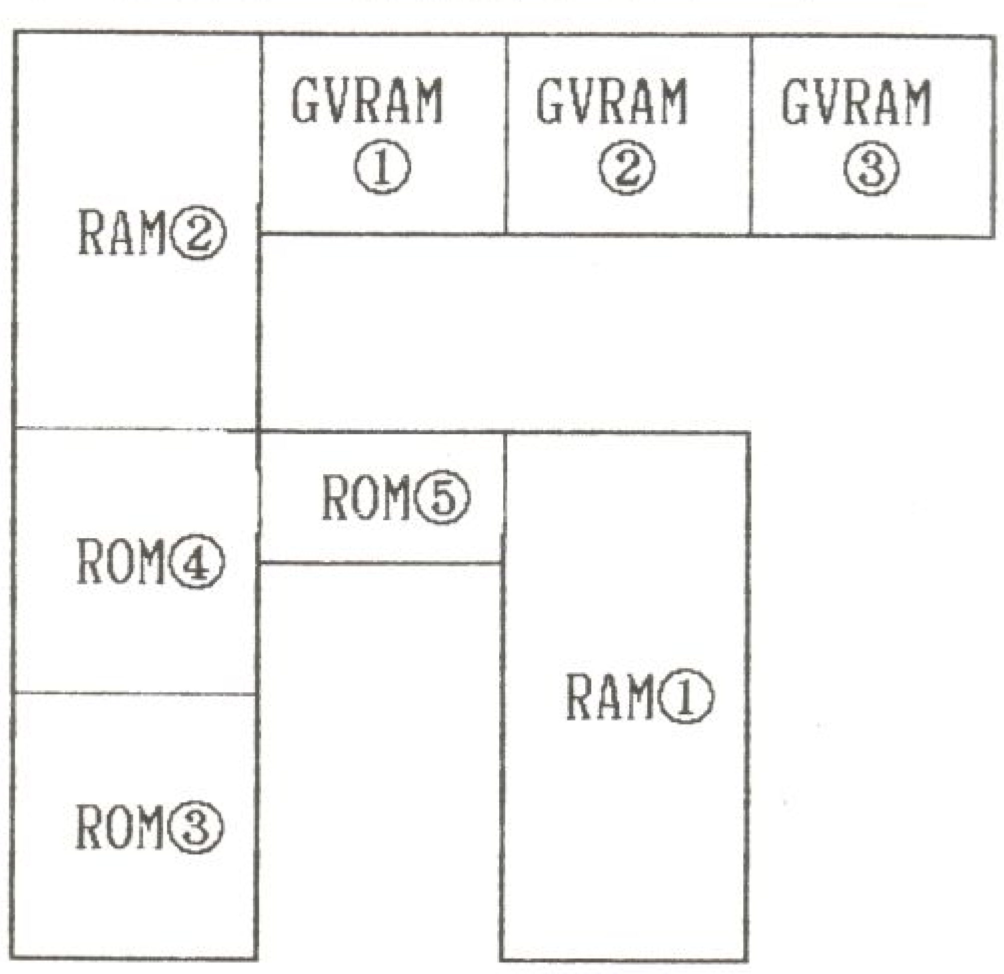


ＣＰＵは６４ＫＢのアドレス空間を持つμＰＤ７８０を使用している。
大容量のメモリーを接続するため、同一アドレス空間に複数のメモリーを割り当てる、バンク切り替えの手法を採用している。

（注１）増設ＲＯＭ　……… ８ＫＢ単位で、最大７バンクまで増設可能。
（注２）増設ＲＡＭ　……… ３２ＫＢ単位で、バンク切り替えをしてアクセスする。

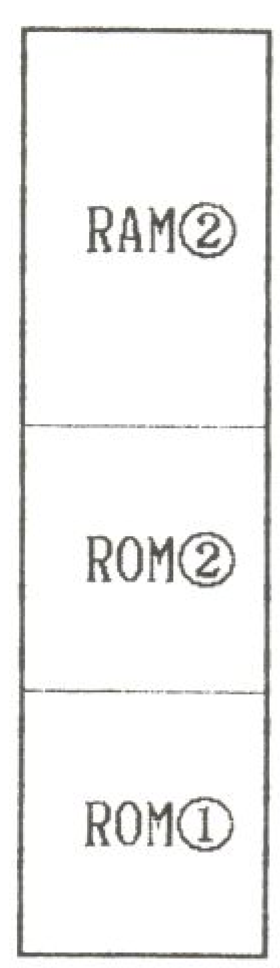
ＰＣ－８８０１には上の図から、動作モードに応じて３種類のメモリー割り当て方法がある。

(1)　Ｎ８８－ＢＡＳＩＣモード



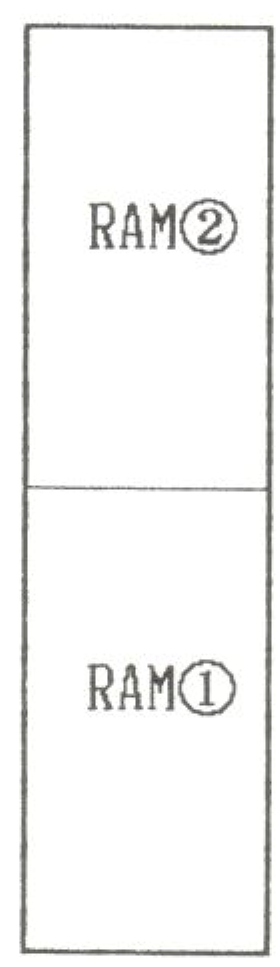
ＰＣ－８８０１の標準ＢＡＳＩＣを動作させるモード。次頁をみよ

(2)　Ｎ－ＢＡＳＩＣ



ＰＣ－８００１とのＢＡＳＩＣ互換モード

(3)　ＲＡＭモード



他のシステムを動作させるモード

☆N８８－ＢＡＳＩＣモードのメモリーアドレスマップ☆　前頁（１）の詳細説明

アドレス	RAM②	ポインター
FFFF		
FFF8	空きエリア	
F3C8	テキストＶＲＡＭ	
	空きエリア	
F300	割り込みベクトル	
E600	ワークエリア	
	機械語エリア	← 注３ (E7E8)
	スタックエリア（ＣＬＥＡＲ文の第３パラメーターでサイズ指定可）	← (E654) ＢＡＳＩＣで使用するメモリーの上限番地。ＣＬＥＡＲ文の第２パラメーターで変更可能。
	⇩ 文字列エリア	← (EACC)
	フリースペース（ＦＲＥ（０）で確認可能）	← (EAF1) (EB1F) ←
	⇧ 配列変数エリア	(EB1D) ←
	⇧ 単純変数エリア	(EB1B) ←
	ラベルテーブル	(EB16) ←
	ファイルバッファ	
	注４　ドライブテーブル	
(8400)	ファイルポインター	(EC7F) ←
	注４　ドライブポインター	(EC81) ←
8400	注４　ディスクコード	
	テキストウインドー	注５（１ＫＢ）
8000		

アドレス	ＧＶＲＡＭ①	ＧＶＲＡＭ②	ＧＶＲＡＭ③
FFFF–C000	ＧＶＲＡＭ①	ＧＶＲＡＭ②	ＧＶＲＡＭ③

アドレス	ROM	
7FFF	ＲＯＭ④ Ｎ８８－ＢＡＳＩＣ	ＲＯＭ⑤ Ｎ８８－ＢＡＳＩＣ
6000		
4000	ＲＯＭ③ Ｎ８８－ＢＡＳＩＣ	
0000		

ＲＡＭ①	
Ｎ８８－ＢＡＳＩＣ テキスト エリア	空きエリア ⇧
ＸＸ００（オフセット）	ＢＡＳＩＣ プログラム

（注３）
（ポインターアドレス）
エリアの下限または上限番地が格納されている。

（注４）
N88-DISK BASICのときだけ入る。N88-ROM BASICでは 8400Hからファイルポインターが入る。

（注５）
N88-BASIC のテキストエリアは、通常のメモリーマップ上には存在しない。特別なハードウエアにより、テキストウインドーに１ＫＢの範囲を写し出し、読み書きを可能にしている。

テキストウインドーに
写し出されるテキストエリア

テキストウインドー	テキストエリア
８３ＦＦ	ＸＸ００＋０３ＦＦ
⋮	⋮
８０００	ＸＸ００＋００００

ＸＸ００：オフセットアドレス
Ｉ／Ｏポートの７０Ｈに１６進の上2桁（００Ｈ ～７ＦＨ）を設定する。

5．8801　ビデオRAM

ビデオＲＡＭには、画面に表示する文字やグラフィックに関する情報がはいる。ビデオＲＡＭは、テキスト用とグラフィック用とに分かれる。

☆テキスト用ビデオＲＡＭ（ＴＶＲＡＭ）☆

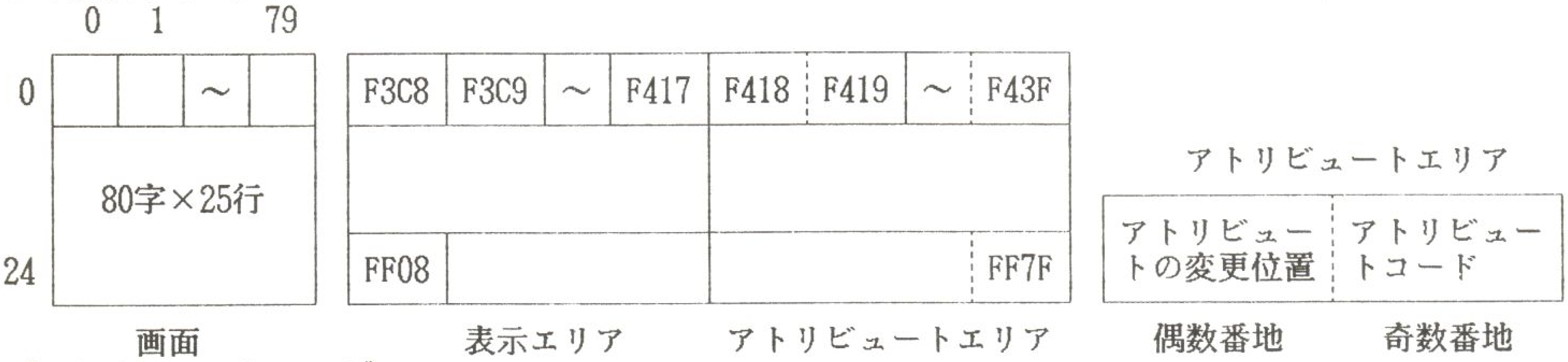
テキスト用ビデオＲＡＭは、テキスト画面に対応する表示エリアと画面上の文字の属性（色、機能）を示すアトリビュートエリアからなる。テキスト画面には漢字は表示できない。

《表示エリア》

テキスト用ビデオＲＡＭは、Ｆ３Ｃ８Ｈ番地からＦＦ７ＦＨ番地までの３ＫＢ。画面１行８０字について８０バイトの表示エリアと４０バイトのアトリビュートエリアを割り当てる。画面に表示する文字の文字コードが、対応する表示エリアにはいる。４０文字モードのときは、表示エリアの偶数番地しか使わない。

《アトリビュートエリア》

２バイトのアトリビュートエリアで文字の属性を示す。上位バイト（偶数番地）にアトリビュートの変更位置がはいり、下位バイト（奇数番地）にアトリビュートコードがはいる。偶数番地で直前の変更位置から何文字目まで属性が同じかを示す。アトリビュートの変更は２０回しかできない。２０回以上変更すると正しく表示しない。



《アトリビュートコード》

アトリビュートコードｂ７～ｂ０のビット値で色や機能を設定する。

b7	b6	b5	b4	b3	b2	b1	b0

カラーモードと白黒モードでは、アトリビュートコードの各ビットの働きが異なる。

《カラーモード》

ＣＯＮＳＯＬＥの第４パラメーターが１のときカラーモードになる。

①色指定

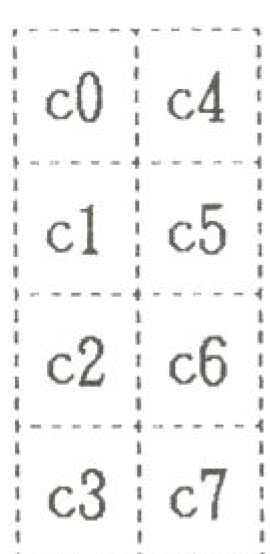
ｂ３を１にして、ｂ７～ｂ５の３ビットで色を指定する。カラーモードのときｂ２～ｂ０は使わない。

色	b7	b6	b5
黒	0	0	0
青	0	0	1
赤	0	1	0
紫	0	1	1
緑	1	0	0
水色	1	0	1
黄	1	1	0
白	1	1	1

（表示エリアのビットパターン）

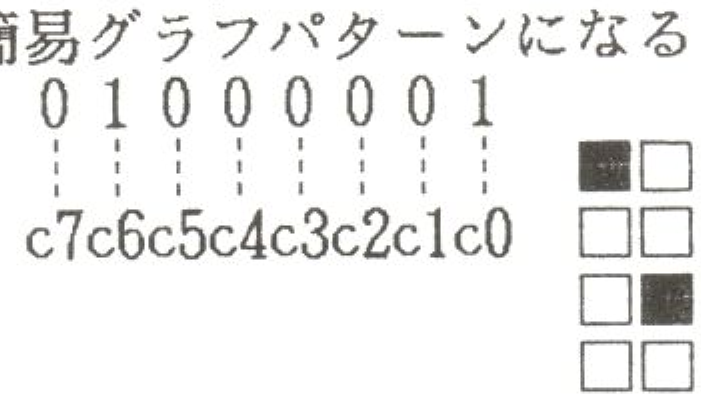
c7	c6	c5	c4	c3	c2	c1	c0

簡易グラフパターン



（例）

『Ａ』の文字コード41Ｈをビットパターンとすると下の簡易グラフパターンになる



（アトリビュートコードの設定）

ｂ４＝０　対応する文字（『キャラクターコード表』参照）

（例）　A

ｂ４＝１　簡易グラフパターン

（例）



②簡易グラフパターン

１文字の字型を８分割して、８ビットのビットパターンで表した文字を簡易グラフパターンという。簡易グラフパターンを画面に表示するには、表示エリアにビットをセットしてアトリビュートコードのｂ４を１にする。

（例）カラーモードで画面右上に『Ａ』とかいて、アトリビュートエリアのＦ４１９Ｈ番地に５８Ｈをいれると右上に赤色で「▀▄」を表示する。



③属性指定
ｂ３を０にして、ｂ５、ｂ４、ｂ２、ｂ１、ｂ０の５ビットで属性を表す。ｂ７、ｂ６は使わない。

ビット値	b7	b6	b5	b4	b3	b2	b1	b0
0	×	×	ノーマル	ノーマル	0	ノーマル	ノーマル	ノーマル
1	×	×	アンダーライン	アッパーライン	0	リバース	ブリンク	シークレット

《白黒モード》
ＣＯＮＳＯＬＥの第４パラメーターを０にすると白黒モードになる。白黒モードのとき、ｂ６とｂ３は使わない。

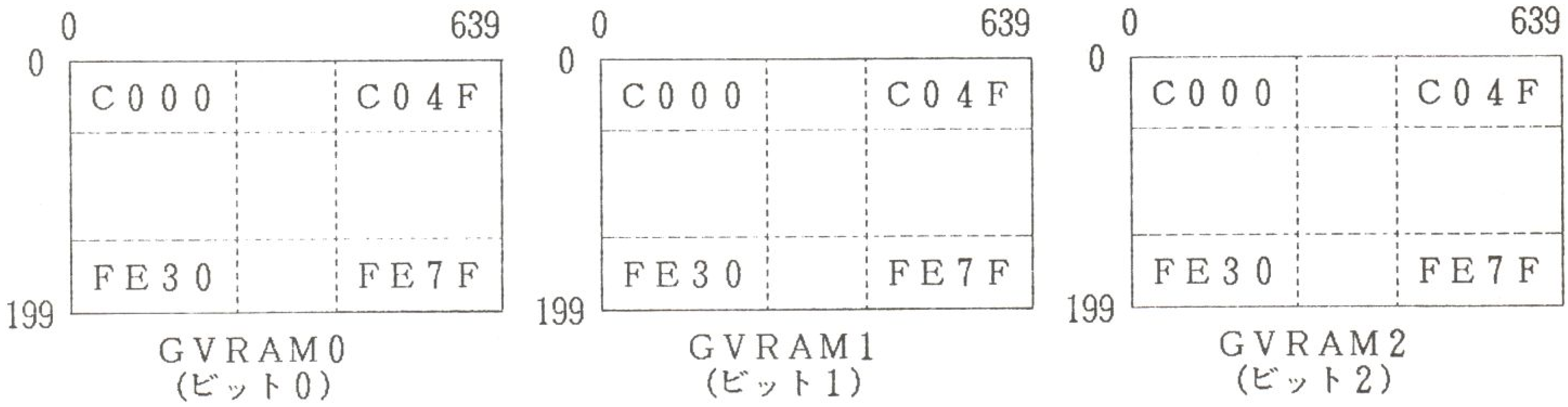
ビット値	b7	b5	b4	b2	b1	b0
0	文字モード	ノーマル	ノーマル	ノーマル	ノーマル	ノーマル
1	簡易グラフ	アンダーライン	アッパーライン	リバース	ブリンク	シークレット

（例）
画面の１行目に対応するアトリビュートエリアのＦ４１８Ｈ番地に８０Ｈ、Ｆ４１９Ｈ番地に３０Ｈをいれると、アンダーラインとアッパーラインを組み合わせて１行目に上下の線が引ける。

	b7	b6	b5	b4	b3	b2	b1	b0	b7	b6	b5	b4	b3	b2	b1	b0	
F418H番地	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	F419H番地

F419H番地 b4 ……アッパーライン
F419H番地 b5 ……アンダーライン

☆グラフィック用ビデオＲＡＭ（ＧＶＲＡＭ）☆



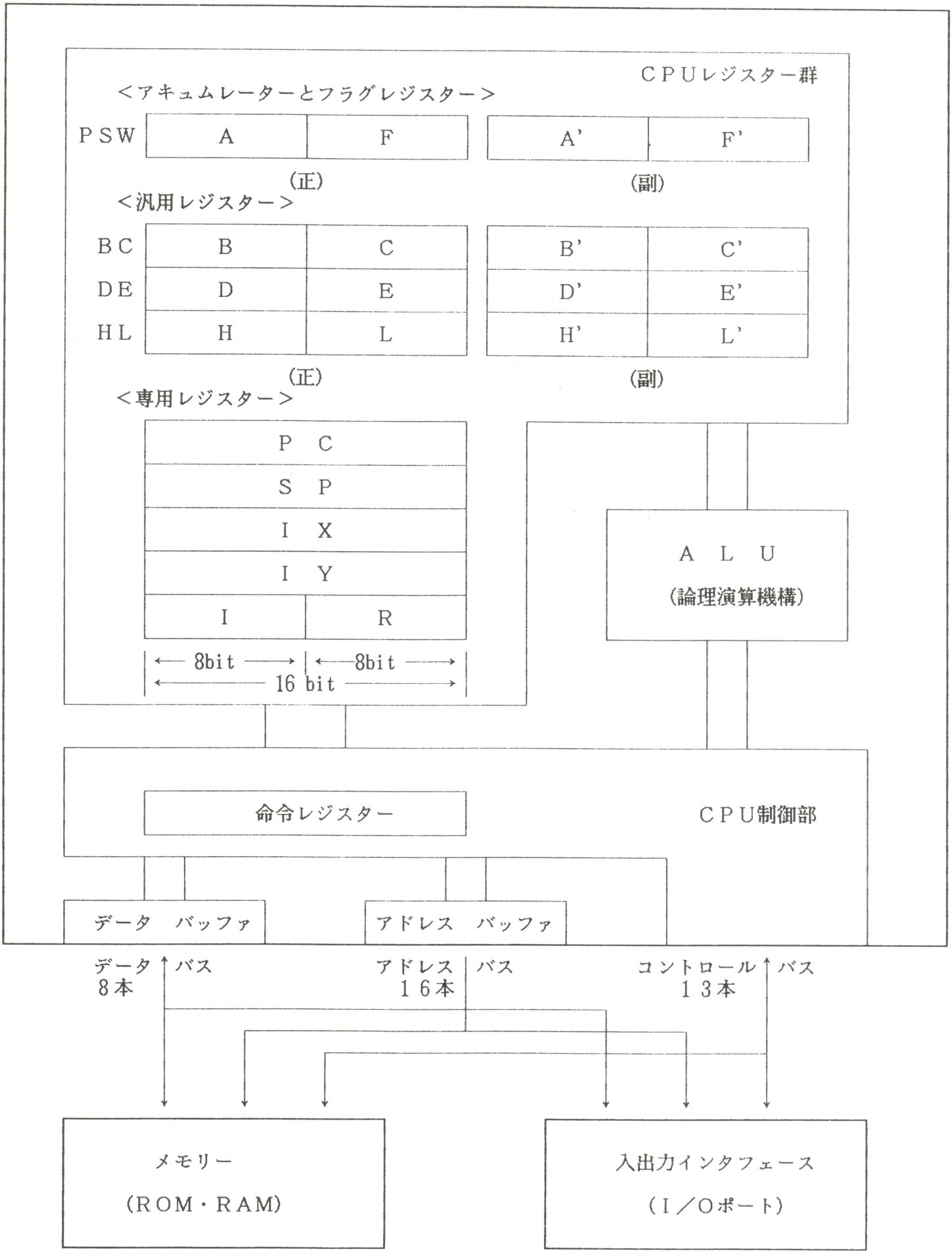
グラフィック用ビデオＲＡＭは、図の３組のＧＶＲＡＭからできている。ＳＣＲＥＥＮ文の第一パラメーターの値によって割り当て方法がかわる。グラフィック用ビデオＲＡＭにアクセスするにはバンクの切り替えが必要である。

パレット番号	b2	b1	b0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

《カラーモード（ＳＣＲＥＥＮの第一パラメーターが０）》
ＧＶＲＡＭ０、１、２をそれぞれビット０、１、２に対応させてパレット番号とし、６４０×２００ドットのカラー画面１ページを構成する。３つのＧＶＲＡＭで各ドットのパレット番号を表す。パレット番号は左表のとおり。
《白黒モード（ＳＣＲＥＥＮの第一パラメーターが１）》
ＧＶＲＡＭ０、１、２を独立した三枚のビデオＲＡＭとして使用する。６４０×２００ドットの白黒画面３ページを自由に組み合わせて表示できる。

《高分解能白黒モード（ＳＣＲＥＥＮの第一パラメーターが２）》
ＧＶＲＡＭ０を画面の上半分、ＧＶＲＡＭ１を画面の下半分として使用し、６４０×４００ドットの白黒画面１ページを構成する。専用高解像度ディスプレーが必要。

6．8801　CPU基本ブロック図



☆アキュムレーターとフラグレジスター☆

アキュムレーターとフラグレジスターは、（Ａ，Ｆ）と（Ａ’，Ｆ’）の正副２組ある。
正副のレジスターEXchange命令で切り替える。

Ａ（Ａ’）…アキュムレーター
演算や比較などのデータ処理に使うレジスター。算術、論理演算は、Ａレジスターと、他のレジスター（Ｂ・Ｃ・Ｄ・Ｅ・Ｈ・Ｌ・Ａ）または、メモリー、あるいはイミディエートデータとの間でなされ、結果はＡレジスターに入る。
Ｆ（Ｆ’）……フラグレジスター
演算の結果生じた状態を、ビットが０か１かで保持する。

b7	b6	b5	b4	b3	b2	b1	b0
S	Z	X	H	X	P／V	N	C

Ｓ　　：サインフラグ……………………結果が負のときセットされる
Ｚ　　：ゼロフラグ………………………結果が０のときセットされる
Ｈ　　：ハーフキャリーフラグ…………ｂ３からｂ４への桁上がりがあったときセットされる
Ｐ／Ｖ：パリティ／オーバフロー………結果の８ビットに１ビットが偶数個含まれるときや、演算結果がオーバーフローしたときセットされる
Ｎ　　：減算フラグ………………………減算命令を実行したときセットされる
Ｃ　　：キャリーフラグ…………………最上位ビット（ｂ７）から桁上がりが生じたときセットされる

ＰＳＷ：アキュムレーターとフラグレジスターのペアレジスターをプログラムステータスワードと呼ぶ。〔ＡＦ（ＡＦ’）〕

☆汎用レジスター☆

Ｂ・Ｃ・Ｄ・Ｅ・Ｈ・Ｌの６個のレジスターが正副２組ある。正副のレジスターはEXchange命令で切り替える。ＢとＣ、ＤとＥ、ＨとＬを、１６ビットのペアレジスターとして使うときは、ＢＣ・ＤＥ・ＨＬの名前を使う。
Ｂ（Ｂ’）レジスター……繰り返し処理のループ回数を制御するとき、減数カウンターとして使う。
Ｃ（Ｃ’）レジスター……入出力命令で、入出力のポートアドレスを指定するのに使う。

ＢＣ（ＢＣ’）
ＤＥ（ＤＥ’）
ＨＬ（ＨＬ’）

・１６ビット演算は、ＢＣ・ＤＥ・ＨＬのペアレジスターとＳＰ・ＩＸ　ＩＹの各レジスター間でなされる。
・レジスター間接アドレッシングで、アドレス指定レジスターとして使われる。
・ブロック転送命令では、ＢＣレジスターにデータのバイト数、ＨＬレジスターにデータのアドレスを、ＤＥレジスターに転送先のアドレスを指定する。

☆専用レジスター☆

ＰＣ…プログラムカウンター
インストラクションを逐次処理するためのアドレスカウンター。命令の読み出しにともない、カウンターの値は自動的に変更され、次に実行する命令のアドレスを常に保持する。１６ビットの大きさで、０～６５５３５番地にある命令を指示できる。
ＳＰ…スタックポインター
メモリーを、プッシュ・ダウン・スタック（後入れ先出し棚）として利用するためのレジスター。スタックはメモリーの上位番地から作られるため、ポインターにはスタックの先頭（最下位）アドレスが保持される。スタックには、サブルーチンの呼び出しや割り込み発生時に、プログラムカウンター、各レジスターの内容を退避する。
ＩＸ，ＩＹインデックス・レジスター
インデックスアドレッシングでオペランドアドレスを修飾するとき、ベースアドレスを提供する。ＩＸとＩＹの２つある。
Ｉ…インタラプト・ページ・アドレス・レジスター
モード２の割り込み発生時に、割り込み処理ルーチンの飛び先テーブルのアドレス上位８ビットとして使われる。割り込みテーブル・アドレスの下位８ビットは、入出力インタフェースよりデータバスを通して読み込まれる。
Ｒ…メモリー・リフレッシュレジスター
メモリーにＤ－ＲＡＭを使用する場合、記憶を保持するメモリーリフレッシュ動作で、リフレッシュアドレスのカウンターとして使われる。

7. 8801　機械語の説明

ここでは機械語を機能別に分類し、ニーモニックの表記方法を一覧表にまとめた。機械語を初めて学ぶ読者は、まず「ＵＳＲ関数とＣＡＬＬ文の使い方」でＢＡＳＩＣと機械語の関係をつかみ、「機械語プログラム集」をためしてからこの章を読むとよい。

一覧表は、μＰＤ７８０の命令を機能別に大きく分類してある。モニターのアセンブラーで使えるものはニーモニックで表記してある。モニターのアセンブラーでは使えない拡張命令は、機械語を直接モニターのＳコマンドかＥコマンドで入れれば実行できるため、機械語で説明した。「モニターの使い方」参照。

☆表の読み方☆

ニーモニック	長さ	フラグ ＳＺＨＰＮＣ	説　明
《転送命令》レジスターと、レジスターやメモリーの間でデータを移動する			
ＭＯＶ　reg1,reg2	1	−−−−−−−	MOVe　reg2の内容をreg1に移す reg1、reg2は８ビットレジスター。ＭならＨＬで指したメモリーをレジスターとみなす

説明：ニーモニックのもとになった英語と、命令の説明

フラグ：命令の実行によるフラグの変化
- Ｓ：セットされる　（1になる）
- Ｒ：リセットされる（0になる）
- Ｘ：不定（意味がない）
- Ｏ：演算の結果によって変化する
- −：変化しない

長さ：命令の長さ
- １〜４バイトの命令がある

reg2：ソースオペランド

reg1：デスティネーションオペランド
- 操作の対象を表す

ＭＯＶ：オペレーションコード
- 操作を表す

オペランドの表現

reg …………	レジスター	Ｈ、Ｌ、Ａ、Ｂ、Ｃ、Ｄ、Ｅ、Ｍ（HLの指すメモリー）
pair …………	ペアレジスター	Ｂと指定すればＢＣペアレジスターになる。したがってＤはＤＥ、ＨはＨＬの意味になる。スタックポインターはＳＰと指定し、プログラム・ステータス・ワード（Ａレジスターとフレジスター）はＰＳＷと指定する。
data …………	１バイトのデータ	００Ｈ〜ＦＦＨ
wdata …………	２バイトのデータ	００００Ｈ〜ＦＦＦＦＨ
adr …………	番地	２バイトで番地を表す。ただし、相対分岐命令に限り、オペランドに指定したとび先番地は、ＰＣ（プログラムカウンター）に対する１バイトの変位（相対番地）になる。したがって、とび先はＰＣに対して−１２８バイト〜＋１２８バイトの範囲になければならない。ＰＣは次の命令の番地を指している点に注意
port …………	入出力ポート番号	１バイトの番号。００Ｈ〜ＦＦＨ。「Ｉ／Ｏアドレスマップ」参照。

ニーモニック	長さ	フラグ SZHPNC	説　明
《転送命令》レジスターと、レジスターやメモリーの間でデータを移動する			
MOV　reg1,reg2	1	ーーーーーー	MOVe　reg2の内容をreg1に移す reg1、reg2は8ビットレジスター。MならHLで指したメモリーをレジスターとみなす
MVI　reg,data	2	ーーーーーー	MoVe Immediate　1バイトの定数を8ビットレジスターに入れる
LDA　adr	3	ーーーーーー	LoaD A direct　adrで示したメモリーの内容をAレジスターに入れる。機械語ではadr の上位と下位が逆転する
STA　adr	3	ーーーーーー	Store A direct　Aレジスターの内容をadr で示すメモリーに入れる
LDAX　pair	1	ーーーーーー	LoaD A indeX　ペアレジスターBCまたはDEで指したメモリーの内容をAレジスターに入れる
STAX　pair	1	ーーーーーー	STore A indeX　Aレジスターの内容をペアレジスターBCまたはDEで示すメモリーに入れる
LXI　pair,wdata	3	ーーーーーー	Load indeX Immediate　2バイトの定数をペアレジスターに入れる。HL、BC、DE、SPが指定できる
LHLD　adr	3	ーーーーーー	Load HL Direct　adrで示すメモリーの内容をHLレジスターに入れる
SHLD　adr	3	ーーーーーー	Store HL Direct　HLレジスターの内容をadr で示すメモリーに入れる
XCHG	1	ーーーーーー	eXCHanGe　ペアレジスターHLとDEの内容を交換する
XTHL	1	ーーーーーー	eXchange sTack with HL　スタックの先頭とHLレジスターの内容を交換する
SPHL	1	ーーーーーー	set SP to HL　HLレジスターの内容をスタックポインターに入れる
PUSH　pair	1	ーーーーーー	PUSH　ペアレジスターの内容をスタックに入れる HL、BC、DE、PSW（FとA）が指定できる
POP　pair	1	PSWのときすべてかわる	POP　スタックの内容をペアレジスターに入れる HL、BC、DE、PSW（FとA）が指定できる
《入出力命令》　Aレジスターと入出力ポートとのやりとりをする ポート番号には00H～FFHが指定できる。次の二つの命令で入出力装置の制御やデータの受け渡しをする			
IN　port	2	ーーーーーー	INput　portで示すポートから1バイトのデータをAレジスターに取り込む
OUT　port	2	ーーーーーー	OUTput　Aレジスターの内容をportで示すポートに出す

ニーモニック	長さ	フラグ ＳＺＨＰＮＣ	説　　　　　明
《演算命令》　算術・論理演算をする。結果はＡレジスターに入る。フラグの変化に注意			
ＩＮＲ　　reg	１	ＯＯＯＯＲ－	INcrement Register　regで指定したレジスターの内容を１増やす
ＤＣＲ　　reg	１	ＯＯＯＯＳ－	DeCrement Register　regで指定したレジスターの内容を１減らす
ＩＮＸ　　pair	１	－－－－－－	INcrement indeX　ペアレジスターの内容を１増やす　ＨＬ、ＢＣ、ＤＥ、ＳＰが指定できる
ＤＣＸ　　pair	１	－－－－－－	DeCrement indeX　ペアレジスターの内容を１減らす　ＨＬ、ＢＣ、ＤＥ、ＳＰが指定できる
ＣＭＡ	１	－－Ｓ－Ｓ－	CoMplement A　Ａレジスターの内容を反転する
ＤＡＡ	１	ＯＯＯＯ－Ｏ	Decimal Adjust A　ＢＣＤ（２進化１０進数）を直接計算する命令はないので、Ａレジスターにある２進数計算の結果を、この命令でＢＣＤにもどす
ＲＡＬ	１	－－Ｒ－ＲＯ	Rotate A Left　ＡレジスターとＣフラグ（キャリーフラグ、桁上がり）をつなげ、左にまわす。最上位ビットはＣフラグに入り、Ｃフラグは最下位ビットに入る
ＲＡＲ	１	－－Ｒ－ＲＯ	Rotate A Right　ＡレジスターとＣフラグをつなげ右にまわす。最下位ビットはＣフラグに入り、Ｃフラグは最上位ビットに入る
ＲＬＣ	１	－－Ｒ－ＲＯ	Rotate Left Circuler　Ａレジスターの内容を左にまわす。最上位ビットはＣフラグと最下位ビットに入る
ＲＲＣ	１	－－Ｒ－ＲＯ	Rotate Right Circuler　Ａレジスターの内容を右にまわす。最下位ビットはＣフラグと最上位ビットに入る
ＡＤＤ　　reg	１	ＯＯＯＯＲＯ	ADD　reg で指定したレジスターの内容をＡレジスターに足す
ＡＤＩ　　data	２	ＯＯＯＯＲＯ	ADd Immediate　１バイト定数をＡレジスターに足す
ＡＤＣ　　reg	１	ＯＯＯＯＲＯ	ADd with Carry　regで指定したレジスターの内容をＣフラグとともにＡレジスターに足す
ＡＣＩ　　data	２	ＯＯＯＯＲＯ	Add with Carry Immediate　１バイト定数をＣフラグとともにＡレジスターに足す
ＳＵＢ　　reg	１	ＯＯＯＯＳＯ	SUbtract　regで指定したレジスターの内容をＡレジスターから引く
ＳＵＩ　　data	２	ＯＯＯＯＳＯ	SUbtract Immediate　１バイト定数をＡレジスターから引く
ＳＢＢ　　reg	１	ＯＯＯＯＳＯ	SuBtract with Borrow　regで指定したレジスターの内容をＣフラグとともにＡレジスターから引く
ＳＢＩ　　data	２	ＯＯＯＯＳＯ	SuBtract with Borrow Immediate　１バイト定数をＣフラグとともにＡレジスターから引く

ニーモニック	長さ	フラグ SZHPNC	説　　　　明
ANA　reg	1	OOSORR	ANd A　regで指定したレジスターの内容とAレジスターの内容の論理積をとる
ANI　data	2	OOSORR	ANd a Immediate　１バイト定数とAレジスターの内容の論理積をとる
XRA　reg	1	OORORR	eXclusive oR A　regで指定したレジスターの内容とAレジスター内容の排他的論理和をとる
XRI　data	2	OORORR	eXclusive oR Immediate　１バイト定数とAレジスターの内容の排他的論理和をとる
ORA　reg	1	OORORR	OR A　regで指定したレジスターの内容とAレジスターの内容の論理和をとる
ORI　data	2	OORORR	OR a Immediate　１バイト定数とAレジスターの内容の論理和をとる
CMP　reg	1	OOOOSO	CoMPare　比較命令。regで指定したレジスターの内容をAレジスターから引く。Aレジスターの内容はかわらず、フラグのみセットする
CPI　data	2	OOOOSO	ComPare Immediate　１バイト定数をAレジスターから引く。Aレジスターの内容はかわらずフラグのみセット
STC	1	−−R−RS	SeT Carry　Cフラグをセットする
CMC	1	−−X−RO	CoMplement Carry　Cフラグを反転する
DAD　pair	1	−−X−RO	Double ADd　pairで指定したペアレジスターの内容をHLレジスターに足す。SPの値はこの命令で取り出す BC、DE、HL、SPが指定できる
《分岐命令》　プログラムの実行の順番をかえる命令			
PCHL	1	−−−−−−−	set PC to HL　ペアレジスターHLの内容をPC（プログラムカウンター）に入れる。HLの指す番地へとばす命令。HLの内容はかわらない
RST　n	1	−−−−−−−	ReSTart　nは0から7の8種。それぞれ対応する番地をサブルーチンとして呼び出す。戻り番地はスタックに積まれる。nに8をかけた番地からサブルーチンを入れておく 例　RST　7　の場合0038H番地から始まる
JMP　adr	3	−−−−−−−	JuMP　adrで示す番地にとぶ とび先の番地は2バイトデータと同様、上位バイトと下位バイトを入れ替えたかたちになる
CALL　adr	3	−−−−−−−	CALL　adrで示す番地から始まるサブルーチンを呼び出す。戻り番地はスタックに入る
RET	1	−−−−−−−	RETurn　サブルーチンから復帰。スタックの先頭の示す番地（CALLの次の命令）へ帰る

ニーモニック	長さ	フラグ SZHPNC	説　　明
《条件分岐命令》　フラグの状態をテストして、条件が成り立てば分岐する　分岐命令の機能やオペランドの指定は、それぞれJMP、CALL、RETに対応する			
JC　adr JNC　adr JZ　adr JNZ　adr JM　adr JP　adr JPE　adr JPO　adr	3	-------	Jump on Carry　Cフラグがセットで分岐 Jump on No Carry　Cフラグがリセットで分岐 Jump on Zero　Zフラグがセットで分岐 Jump on No Zero　Zフラグがリセットで分岐 Jump on Minus　Sフラグがセットで分岐 Jump on Positive　Sフラグがリセットで分岐 Jump on Parity Even　Pフラグがセットで分岐 Jump on Parity Odd　Pフラグがリセットで分岐
CC　adr CNC　adr CZ　adr CNZ　adr CM　adr CP　adr CPE　adr CPO　adr	3	-------	Call on Carry　Cフラグがセットで分岐 Call on No Carry　Cフラグがリセットで分岐 Call on Zero　Zフラグがセットで分岐 Call on No Zero　Zフラグがリセットで分岐 Call on Minus　Sフラグがセットで分岐 Call on Positive　Sフラグがリセットで分岐 Call on Parity Even　Pフラグがセットで分岐 Call on Parity Odd　Pフラグがリセットで分岐
RC RNC RZ RNZ RM RP RPE RPO	1	-------	Return on Carry　Cフラグがセットで分岐 Return on No Carry　Cフラグがリセットで分岐 Return on Zero　Zフラグがセットで分岐 Return on No Zero　Zフラグがリセットで分岐 Return on Minus　Sフラグがセットで分岐 Return on Positive　Sフラグがリセットで分岐 Return on Parity Even　Pフラグがセットで分岐 Return on Parity Odd　Pフラグがリセットで分岐
《相対分岐命令》　この命令の次の番地を0として－128～＋127の範囲にとぶ。拡張命令			
JMPR　adr	2	-------	JuMP Relative　adrは上記の範囲のとび先番地
JRC　adr	2	-------	Jump Relative on Carry　Cフラグがセットなら分岐
JRNC　adr	2	-------	Jump Relative on No Carry　Cフラグがリセットなら分岐
JRZ　adr	2	-------	Jump Relative on Zero　Zフラグがセットなら分岐
JRNZ　adr	2	-------	Jump Relative on No Zero　Zフラグがリセットで分岐
DJNZ　adr	2	-------	Decrement and Jump on No Zero DCR　BとJRNZの合成。Bレジスターの内容を1減らし、結果が0でなければ分岐。0ならつぎの命令へ。ループに使う
《その他の命令》			
EI	1	-------	Enable Interrupts　割り込みを許す
DI	1	-------	Disable Interrupts　割り込みを禁じる
HLT	1	-------	HaLT　実行を止める
NOP	1	-------	No OPeration　何もせず次の命令に進む

《拡張命令の機械語表》

モニターのアセンブラーでは使えないが、直接機械語で入力すれば使える拡張命令がある。拡張命令については、次のページ以降にある「拡張命令の機械語表」と下記の「付表①〜⑧」を使って、機械語をひく。手順は、次のとおり。

・「拡張命令の機械語表」から、実行したい機能の機械語をひく
・表中の機械語の欄に○で囲んだ数字（①など）がある場合、数字が示す「付表」を使って機械語を完成させる

（例１）ペアレジスターの内容とＣフラグをＨＬレジスターに足す命令の機械語を探す場合
・「拡張命令の機械語表」の説明から機械語をひくとＥＤ①Ａとある。①の部分については、付表①をみる
・付表①はペアレジスターの指定なので、ＤＥレジスターを使うときは、「５」になる
・したがって機械語は、ＥＡ①Ａの①の部分に５を入れて「ＥＡ５Ａ」になる

（例２）Ｅレジスターのビット２をしらべる命令の機械語を探す場合
・「拡張命令の機械語表」の説明から機械語をひくとＣＢ⑥とある
⑥の部分については　付表⑥をみる
・付表⑥はビットの指定なので、ビット２のときは「５」になる
・付表⑥の参照付表に④とあるので、さらに付表④をみる
・付表④はレジスターの指定なので、Ｅレジスターのときは「３」
・したがって機械語は、付表⑥と④を合わせて、「ＣＢ５３」になる

付表①ペアレジスターの指定

BC	DE	HL	SP
4	5	6	7

付表②・③レジスターの指定

付表	B	C	D	E	H	L	A
②	40	48	50	58	60	68	78
③	41	49	51	59	61	69	79

付表④・⑤レジスターの指定

付表	B	C	D	E	H	L	M	A
④	0	1	2	3	4	5	6	7
⑤	8	9	A	B	C	D	E	F

付表⑥・⑦・⑧ビットの指定

付表	b7	b6	b5	b4	b3	b2	b1	b0
⑥	7	7	6	6	5	5	4	4
⑦	F	F	E	E	D	D	C	C
⑧	B	B	A	A	9	9	8	8
参照付表	⑤	④	⑤	④	⑤	④	⑤	④

《インデックスレジスターを使う拡張命令》

ペアレジスターＨＬを使う機械語の命令の頭にＤＤかＦＤ（１６進数）を付けると、ＨＬレジスターのかわりにインデックスレジスター（ＩＸまたはＩＹレジスター）を使う拡張命令になる。ＤＤを付けた場合、ＩＸレジスターが使われ、ＦＤを付けるとＩＹレジスターが使われる。機械語については、「（８８）ニーモニック機械語の対応表」と「拡張命令の機械語表」参照。

・オペランドにペアレジスターＨＬの意味のＨが書ける命令は、ＨＬのかわりにインデックスレジスターが使われる

（例１）

ニーモニック	機械語	意味
ＩＮＸ　Ｈ	２３	ＨＬレジスターの内容を１増やす
	ＤＤ２３	ＩＸレジスターの内容を１増やす

・オペランドにＭが書ける命令は、ＨＬの指す番地のかわりに、インデックスレジスターの内容に１バイト定数ｎｎの値を加えた番地が対象となる

（例２）

ニーモニック	機械語	意味
ＭＯＶ　Ｄ，Ｍ	５６	ＤレジスターにＨＬレジスターが指すメモリーの内容を入れる
	ＦＤ５６ｎｎ	ＤレジスターにＩＹレジスターの内容＋ｎｎ番地の内容を入れる

・ＣＢＸＸの形をした命令で、かつＨＬレジスターの指す番地を使う命令は、ＨＬの指す番地のかわりにインデックスレジスターの内容に１バイトの定数ｎｎの値を加えた番地が対象となる

（例３）

機械語	意味
ＣＢ５６	ＨＬレジスターの指すメモリーのビット２をテストする
ＤＤＣＢｎｎ５６	ＩＸレジスターの内容にｎｎを足した番地の内容のビット２をテストする

《拡張命令の機械語表》

機械語欄の①～⑧は付表参照

機械語	長さ	フラグ ＳＺＨＰＮＣ	説　　　　明
ＥＤ４４	2	○○○○Ｓ○	NEGate　Ａレジスターの内容を２の補数にする。正なら負、負なら正になる
ＥＤ①Ａ	2	○○×○Ｒ○	ADd with Carry　①で指定したペアレジスターの内容とＣフラグをＨＬレジスターに足す ＢＣ、ＤＥ、ＨＬ、ＳＰが指定できる
ＥＤ①２	2	○○×○Ｓ○	SuBtract with Carry ①で指定したペアレジスターの内容とＣフラグをＨＬレジスターから引く ＢＣ、ＤＥ、ＨＬ、ＳＰが指定できる
ＥＤ②	2	○○Ｒ○Ｒ－	INput　Ｃレジスターの内容をポート番号として②で指定したレジスターに入力する
ＥＤ③	2	－－－－－－	OUTput　Ｃレジスターの内容をポート番号として③で指定したレジスターの内容を出力する
ＣＢ０④	2	○○Ｒ○Ｒ○	Rotate Left Circuler　④で指定したレジスターの内容を左にまわす。最上位ビットはＣフラグと最下位ビットに入る
ＣＢ０⑤	2	○○Ｒ○Ｒ○	Rotate Right Circuler ⑤で指定したレジスターの内容を右にまわす。最下位ビットはＣフラグと最上位ビットに入る
ＣＢ１④	2	○○Ｒ○Ｒ○	Rotate Left ④で指定したレジスターとＣフラグをつなげ左にまわす。最上位ビットはＣフラグに入り、Ｃフラグは最下位ビットに入る
ＣＢ１⑤	2	○○Ｒ○Ｒ○	Rotate Right　⑤で指定したレジスターとＣフラグをつなげ右にまわす。最下位ビットはＣフラグに入り、Ｃフラグは最上位ビットに入る
ＣＢ２④	2	○○Ｒ○Ｒ○	Shift Left Arithmetic　④で指定したレジスターを左にシフトする。最上位ビットはＣフラグに入る。最下位ビットは０になる
ＣＢ２⑤	2	○○Ｒ○Ｒ○	Shift Right Arithmetic　⑤で指定したレジスターを右にシフトする。最下位ビットはＣフラグに入る。最上位ビットは不変
ＣＢ３⑤	2	○○Ｒ○Ｒ○	Shift Right Logical　⑤で指定したレジスターを右にシフトする。最下位ビットはＣフラグに入る。最上位ビットは０になる
ＥＤ６Ｆ	2	○○Ｒ○Ｒ－	Rotate Left Digit　ＨＬが指す番地の内容を４ビット左へシフトする。このときＡレジスターの下４ビットがメモリーの下４ビットに入り、メモリーの上４ビットがＡレジスターの下４ビットに入る
ＥＤ６７	2	○○Ｒ○Ｒ－	Rotate Right Digit　ＨＬが指す番地の内容を４ビット右へシフトする。このときＡレジスターの下４ビットがメモリーの上４ビットに入り、メモリーの下４ビットがＡレジスターの下４ビットに入る

機械語欄の①～⑧は付表参照

機械語	長さ	フラグ SZHPNC	説　明
EDA1	2	XOXOS－	ComPare Increment　Aレジスターの内容とHLが指す番地の内容を比較し、等しかったらZフラグがセットされる。比較後、BCは－1、HLは＋1される BC＝0ならPフラグは0となる
EDB1	2	XOXOS－	ComPare Increment Repeat　同上。Aレジスターの内容とHLが指す番地の内容が等しくなるか、BCが0になるまで繰り返す
EDA9	2	XOXOS－	ComPare Decrement　Aレジスターの内容とHLが指す番地の内容を比較し、等しかったらZフラグがセットされる。比較後、BC、HLとも－1される BC＝0ならPフラグは0となる
EDB9	2	XOXOS－	ComPare Decrement Repeat　同上。Aレジスターの内容とHLが指す番地の内容が等しくなるか、BCが0になるまで繰り返す
EDA0	2	XXROR－	LoaD Increment　HLが指す番地の内容をDEが指す番地にうつす。HL、DEは＋1、BCは－1される ブロック転送命令
EDB0	2	XXRRR－	LoaD Increment Repeat　同上。BCが0になるまで繰り返す
EDA8	2	XXROR－	LoaD Decrement　HLが指す番地の内容をDEが指す番地にうつす。HL、DE、BCとも－1される
EDB8	2	XXRRR－	LoaD Decrement Repeat　同上。BCが0になるまで繰り返す
EDA2	2	XOXXXX	INput Increment　Cレジスターが指すポートからデータを入力し、HLが指す番地へうつす HLは＋1、Bは－1される。ブロック入力命令
EDA3	2	XOXXXX	OUTput Increment　HLが指す番地からCレジスターが指すポートへ出力。HLは＋1、Bレジスターは－1される。ブロック出力命令
EDB2	2	XSXXXX	INput Increment Repeat　ブロック入力命令をBレジスターが0になるまで繰り返す
EDB3	2	XSXXXX	OuTput Increment Repeat　ブロック出力命令をBレジスターが0になるまで繰り返す
EDAA	2	XOXXXX	INput Decrement　HLが－1されるブロック入力命令
EDAB	2	XOXXXX	OUTput Decrement　HLが－1されるブロック出力命令
EDBA	2	XSXXXX	INput Decrement Repeat　HLが－1されるブロック入力命令をBレジスターが0になるまで繰り返す
EDBB	2	XSXXXX	OuTput Decrement Repeat　HLが－1されるブロック出力命令をBレジスターが0になるまで繰り返す

機械語欄の①～⑧は付表参照

機械語	長さ	フラグ SZHPNC	説　　　明
CB⑥	2	XOSXR－	BIT test　⑥で示すレジスターのビットをしらべる　1ならZフラグを0、0なら1とする
CB⑦	2	－－－－－－－	bit SET　⑦で示すレジスターのビットを1にする
CB⑧	2	－－－－－－－	bit RESet　⑧で示すレジスターのビットを0にする
ED4D	2	－－－－－－－	RETurn from Interrupt　割り込み処理ルーチンから戻る
ED45	2	－－－－－－－	RETurn from NMI　NMI（ノンマスカブル割り込み）に対する処理ルーチンから戻る。NMI割り込み処理ルーチンの開始番地は66H
ED46	2	－－－－－－－	Interrupt Mode　割り込みモード0。割り込みを受けつけると、データバスから割り込み命令（CALLやRST）をとりこんで割り込み処理をする
ED56	2	－－－－－－－	Interrupt Mode　割り込みモード1。割り込みを受け付けると38H番地をコールする
ED5E	2	－－－－－－－	Interrupt Mode　割り込みモード2。割り込みを受け付けると割り込みテーブルに入っている番地をコールする　割り込みテーブルの番地の上位8ビットはIレジスターから、下位8ビットはデータバスから取り込み、最下位ビットを0として間接コールする　割り込みモード命令（ED46、ED56、ED5E）は割り込みのモードを決める。割り込みはEI命令で可能になる。割り込みが起きた後は禁止状態になるので、割り込み処理ルーチンから戻る直前に、EI命令で割り込み可能な状態にする
ED57	2	OOROR－	LoaD　Iレジスターの内容をAレジスターに取り出す　Pフラグに割り込みフリップフロップの状態が入る
ED5F	2	OOROR－	LoaD　Rレジスターの内容をAレジスターに取り出す　Pフラグに割り込みフリップフロップの状態が入る
ED47	2	－－－－－－－	LoaD　Aレジスターの内容をIレジスターに入れる
ED4F	2	－－－－－－－	LoaD　Aレジスターの内容をRレジスターに入れる
ED①Bnnnn	4	－－－－－－－	LoaD　nnnn番地の内容を①で指定したペアレジスターに入れる。BC、DE、SPのみ可能
ED①3nnnn	4	－－－－－－－	LoaD　①で指定したペアレジスターの内容をnnnn番地へ入れる。BC、DE、SPのみ可能
08	1	－－－－－－－	EXchange　PSWを副PSWに切り替える
D9	1	－－－－－－－	EXchange indeX　ペアレジスターBC、DE、HLを副レジスターにきりかえる

8. 8801 ニーモニックと機械語の対応表

rはレジスター。nは1バイトのデータまたは相対番地。nnは2バイトのデータまたは番地

8ビット転送・演算

r=	A	B	C	D	E	H	L	M		
MOV A,r	7F	78	79	7A	7B	7C	7D	7E	MVI A,n	3E
MOV B,r	47	40	41	42	43	44	45	46	MVI B,n	06
MOV C,r	4F	48	49	4A	4B	4C	4D	4E	MVI C,n	0E
MOV D,r	57	50	51	52	53	54	55	56	MVI D,n	16
MOV E,r	5F	58	59	5A	5B	5C	5D	5E	MVI E,n	1E
MOV H,r	67	60	61	62	63	64	65	66	MVI H,n	26
MOV L,r	6F	68	69	6A	6B	6C	6D	6E	MVI L,n	2E
MOV M,r	77	70	71	72	73	74	75	--	MVI M,n	36
ADD r	87	80	81	82	83	84	85	86	ADI n	C6
ADC r	8F	88	89	8A	8B	8C	8D	8E	ACI n	CE
SUB r	97	90	91	92	93	94	95	96	SUI n	D6
SBB r	9F	98	99	9A	9B	9C	9D	9E	SBI n	DE
ANA r	A7	A0	A1	A2	A3	A4	A5	A6	ANI n	E6
XRA r	AF	A8	A9	AA	AB	AC	AD	AE	XRI n	EE
ORA r	B7	B0	B1	B2	B3	B4	B5	B6	ORI n	F6
CMP r	BF	B8	B9	BA	BB	BC	BD	BE	CPI n	FE
INR r	3C	04	0C	14	1C	24	2C	34	--------	
DCR r	3D	05	0D	15	1D	25	2D	35	--------	

8ビット転送

LDAX B	0A
LDAX D	1A
STAX B	02
STAX D	12
STA nn	32
LDA nn	3A

入出力

IN n	DB
OUT n	D3

回転

RLC	07
RRC	0F
RAL	17
RAR	1F

操作

DAA	27
CMA	2F
CMC	3F
STC	37

CPU制御

EI	FB
DI	F3
NOP	00
HLT	76

rはレジスター。nは1バイトのデータまたは相対番地。nnは2バイトのデータまたは番地

16ビット転送・演算	r =	PSW	B	D	H	SP
	PUSH r	F5	C5	D5	E5	--
	POP r	F1	C1	D1	E1	--
	DAD r	--	09	19	29	39
	INX r	--	03	13	23	33
	DCX r	--	0B	1B	2B	3B

16ビット転送		
	LXI B, nn	01
	LXI D, nn	11
	LXI H, nn	21
	LXI SP, nn	31
	LHLD nn	2A
	SHLD nn	22

ジャンプ	JMPR n	18
	JRZ n	28
	JRNZ n	20
	JRC n	38
	JRNC n	30
	JMP nn	C3
	JZ nn	CA
	JNZ nn	C2
	JC nn	DA
	JNC nn	D2
	JPE nn	EA
	JPO nn	E2
	JP nn	F2
	JM nn	FA
	DJNZ n	10
リスタート	RST 0	C7
	RST 1	CF
	RST 2	D7
	RST 3	DF
	RST 4	E7
	RST 5	EF
	RST 6	F7
	RST 7	FF

転送	SPHL	F9
	PCHL	E9

交換	XCHG	EB
	XTHL	E3

コール	CALL nn	CD
	CZ nn	CC
	CNZ nn	C4
	CC nn	DC
	CNC nn	D4
	CPE nn	EC
	CPO nn	E4
	CP nn	F4
	CM nn	FC
リターン	RET	C9
	RZ	C8
	RNZ	C0
	RC	D8
	RNC	D0
	RPE	E8
	RPO	E0
	RP	F0
	RM	F8

９．8801　機械語とニーモニックの対応表

ｎは１バイトのデータまたは相対番地。ｎｎは２バイトのデータまたは番地

（００～５０）

機械	ニーモニック	機械	ニーモニック	機械	ニーモニック
００	ＮＯＰ	１Ｂ	ＤＣＸ　Ｄ	３６	ＭＶＩ　Ｍ，ｎ
０１	ＬＸＩ　Ｂ，ｎｎ	１Ｃ	ＩＮＲ　Ｅ	３７	ＳＴＣ
０２	ＳＴＡＸ　Ｂ	１Ｄ	ＤＣＲ　Ｅ	３８	ＪＲＣ　ｎ
０３	ＩＮＸ　Ｂ	１Ｅ	ＭＶＩ　Ｅ，ｎ	３９	ＤＡＤ　ＳＰ
０４	ＩＮＲ　Ｂ	１Ｆ	ＲＡＲ	３Ａ	ＬＤＡ　ｎｎ
０５	ＤＣＲ　Ｂ	２０	ＪＲＮＺ　ｎ	３Ｂ	ＤＣＸ　ＳＰ
０６	ＭＶＩ　Ｂ，ｎ	２１	ＬＸＩ　Ｈ，ｎｎ	３Ｃ	ＩＮＲ　Ａ
０７	ＲＬＣ	２２	ＳＨＬＤ　ｎｎ	３Ｄ	ＤＣＲ　Ａ
０８	------------------------	２３	ＩＮＸ　Ｈ	３Ｅ	ＭＶＩ　Ａ，ｎ
０９	ＤＡＤ　Ｂ	２４	ＩＮＲ　Ｈ	３Ｆ	ＣＭＣ
０Ａ	ＬＤＡＸ　Ｂ	２５	ＤＣＲ　Ｈ	４０	ＭＯＶ　Ｂ，Ｂ
０Ｂ	ＤＣＸ　Ｂ	２６	ＭＶＩ　Ｈ，ｎ	４１	ＭＯＶ　Ｂ，Ｃ
０Ｃ	ＩＮＲ　Ｃ	２７	ＤＡＡ	４２	ＭＯＶ　Ｂ，Ｄ
０Ｄ	ＤＣＲ　Ｃ	２８	ＪＲＺ　ｎ	４３	ＭＯＶ　Ｂ，Ｅ
０Ｅ	ＭＶＩ　Ｃ，ｎ	２９	ＤＡＤ　Ｈ	４４	ＭＯＶ　Ｂ，Ｈ
０Ｆ	ＲＲＣ	２Ａ	ＬＨＬＤ　ｎｎ	４５	ＭＯＶ　Ｂ，Ｌ
１０	ＤＪＮＺ　ｎ	２Ｂ	ＤＣＸ　Ｈ	４６	ＭＯＶ　Ｂ，Ｍ
１１	ＬＸＩ　Ｄ，ｎｎ	２Ｃ	ＩＮＲ　Ｌ	４７	ＭＯＶ　Ｂ，Ａ
１２	ＳＴＡＸ　Ｄ	２Ｄ	ＤＣＲ　Ｌ	４８	ＭＯＶ　Ｃ，Ｂ
１３	ＩＮＸ　Ｄ	２Ｅ	ＭＶＩ　Ｌ，ｎ	４９	ＭＯＶ　Ｃ，Ｃ
１４	ＩＮＲ　Ｄ	２Ｆ	ＣＭＡ	４Ａ	ＭＯＶ　Ｃ，Ｄ
１５	ＤＣＲ　Ｄ	３０	ＪＲＮＣ　ｎ	４Ｂ	ＭＯＶ　Ｃ，Ｅ
１６	ＭＶＩ　Ｄ，ｎ	３１	ＬＸＩ　ＳＰ，ｎｎ	４Ｃ	ＭＯＶ　Ｃ，Ｈ
１７	ＲＡＬ	３２	ＳＴＡ　ｎｎ	４Ｄ	ＭＯＶ　Ｃ，Ｌ
１８	ＪＭＰＲ　ｎ	３３	ＩＮＸ　ＳＰ	４Ｅ	ＭＯＶ　Ｃ，Ｍ
１９	ＤＡＤ　Ｄ	３４	ＩＮＲ　Ｍ	４Ｆ	ＭＯＶ　Ｃ，Ａ
１Ａ	ＬＤＡＸ　Ｄ	３５	ＤＣＲ　Ｍ	５０	ＭＯＶ　Ｄ，Ｂ

(51～AA)

51	MOV D,C	6F	MOV L,A	8D	ADC L
52	MOV D,D	70	MOV M,B	8E	ADC M
53	MOV D,E	71	MOV M,C	8F	ADC A
54	MOV D,H	72	MOV M,D	90	SUB B
55	MOV D,L	73	MOV M,E	91	SUB C
56	MOV D,M	74	MOV M,H	92	SUB D
57	MOV D,A	75	MOV M,L	93	SUB E
58	MOV E,B	76	HLT	94	SUB H
59	MOV E,C	77	MOV M,A	95	SUB L
5A	MOV E,D	78	MOV A,B	96	SUB M
5B	MOV E,E	79	MOV A,C	97	SUB A
5C	MOV E,H	7A	MOV A,D	98	SBB B
5D	MOV E,L	7B	MOV A,E	99	SBB C
5E	MOV E,M	7C	MOV A,H	9A	SBB D
5F	MOV E,A	7D	MOV A,L	9B	SBB E
60	MOV H,B	7E	MOV A,M	9C	SBB H
61	MOV H,C	7F	MOV A,A	9D	SBB L
62	MOV H,D	80	ADD B	9E	SBB M
63	MOV H,E	81	ADD C	9F	SBB A
64	MOV H,H	82	ADD D	A0	ANA B
65	MOV H,L	83	ADD E	A1	ANA C
66	MOV H,M	84	ADD H	A2	ANA D
67	MOV H,A	85	ADD L	A3	ANA E
68	MOV L,B	86	ADD M	A4	ANA H
69	MOV L,C	87	ADD A	A5	ANA L
6A	MOV L,D	88	ADC B	A6	ANA M
6B	MOV L,E	89	ADC C	A7	ANA A
6C	MOV L,H	8A	ADC D	A8	XRA B
6D	MOV L,L	8B	ADC E	A9	XRA C
6E	MOV L,M	8C	ADC H	AA	XRA D

AB	XRA E	C8	RZ	E5	PUSH H
AC	XRA H	C9	RET	E6	ANI n
AD	XRA L	CA	JZ nn	E7	RST 4
AE	XRA M	CB	…………	E8	RPE
AF	XRA A	CC	CZ nn	E9	PCHL
B0	ORA B	CD	CALL nn	EA	JPE nn
B1	ORA C	CE	ACI n	EB	XCHG
B2	ORA D	CF	RST 1	EC	CPE nn
B3	ORA E	D0	RNC	ED	…………
B4	ORA H	D1	POP D	EE	XRI n
B5	ORA L	D2	JNC nn	EF	RST 5
B6	ORA M	D3	OUT n	F0	RP
B7	ORA A	D4	CNC nn	F1	POP PSW
B8	CMP B	D5	PUSH D	F2	JP nn
B9	CMP C	D6	SUI n	F3	DI
BA	CMP D	D7	RST 2	F4	CP nn
BB	CMP E	D8	RC	F5	PUSH PSW
BC	CMP H	D9	…………	F6	ORI n
BD	CMP L	DA	JC nn	F7	RST 6
BE	CMP M	DB	IN n	F8	RM
BF	CMP A	DC	CC nn	F9	SPHL
C0	RNZ	DD	…………	FA	JM nn
C1	POP B	DE	SBI n	FB	EI
C2	JNZ nn	DF	RST 3	FC	CM nn
C3	JMP nn	E0	RPO	FD	…………
C4	CNZ nn	E1	POP H	FE	CPI n
C5	PUSH B	E2	JPO nn	FF	RST 7
C6	ADI n	E3	XTHL		
C7	RST 0	E4	CPO nn		

10．8801　USR関数とCALL文の使い方

☆機械語プログラムの作成とロード☆

《機械語プログラムの作成》
　機械語プログラムは、モニターのコマンドを使って作成する。ディスクやカセットテープに保存しておき必要な時にロードする。また、ＰＯＫＥ文を使い機械語プログラムを直接機械語エリアに書き込む方法もある。

（1）ＣＬＥＡＲ文を使い、機械語プログラムのエリアをメモリーに確保する。
　ＰＣ－８８０１はＥ６００ＨからＦＦＦＦＨ番地をＢＡＳＩＣの作業場所として使っている。
　機械語エリアとして使えるのはＣＬＥＡＲ文で指定した番地からＥ５ＦＦＨ番地までである。

```
CLEAR ,&HE000 ............ E000H番地から機械語エリアを確保する
```

（2）モニターを使い機械語エリアにプログラムを作る。

```
MON .......................... モニターを動かす
h〕AE000 ..................... Aコマンドを使い、E000H番地からアセンブ
                               ルを開始する
E000  21  0000  LXI  H,0 ┐
E003  39        DAD  SP  ├── アセンブラーニーモニックで機械語プログラムを
  :                      ┘    入れる。【RET】で機械語を自動的に表示する
```

　【ＲＥＴ】だけを押すとアセンブル作業は終了する。
　機械語で使うデータはＳコマンドまたはＥコマンドを使い、１６進数でメモリーに書き込む。

（3）機械語プログラムをディスクまたはカセットテープに保存する。

・ディスクに保存する方法

```
h〕^B ................................. モニターからBASICに戻る
OK
BSAVE "PROG0",&HE000,&H0400 ...... 機械語エリアのE000H番地から
OK                                      400Hバイトをプログラムファイ
                                        ル（PROG0）に保存する
```

・カセットテープに保存する方法

```
h〕WPROG0,E000,E3FF .................. 機械語エリアのE000H番地から
                                        E3FFH番地をプログラムファイ
                                        ル（PROG0）に保存する
h〕^B ................................. モニターからBASICに戻る
OK
```

《機械語プログラムのロード》
（1）ディスクの機械語プログラムをロードする方法
　ＢＡＳＩＣの命令だけを使って準備作業ができる。

```
10 CLEAR ,&HE000 ........ 機械語プログラムを作成したときと同じ番地に機
                          械語エリアを確保する
20 BLOAD"PROG0" ......... 機械語エリアにプログラム（PROG0）をロー
                          ドする。
```

（2）カセットテープの機械語プログラムをロードする方法
　モニターのＲコマンドを使ってメモリーにロードする。ＢＡＳＩＣプログラムを実行する前につぎの操作をする。

```
CLEAR ,&HE000 ............ 機械語プログラムを作成したときと同じ番地に機
                           械語エリアを確保する
MON ...................... モニターを動かす
h〕RPROG0 ................ 機械語エリアにプログラム（PROG0）をロー
Found:PROG0                ドする。
h〕^B .................... モニターからBASICに戻る
OK
```

《ＰＯＫＥ文で機械語を扱う方法》

ＰＯＫＥ文を使い機械語プログラムを直接機械語エリアに書き込む。この場合、機械語プログラムをディスクやカセットテープからロードする必要がない。

```
10   CLEAR ,&HE000 ………… 機械語エリアをE000H番地から確保する
20   DATA &H21, &H00 ┐
30   DATA &H00, &H39 ├…… DATA文で機械語プログラムを作る
      ┆              ┘
100  A% = &HE000   ┐
110  FOR I=1 TO nnn ├…… DATA文の機械語プログラムをPOKE文を
120  READ D%         │    使って機械語エリアに書き込む
130  POKE A%, D%     │
140  A% = A%+1       │    nnnは機械語プログラムのバイト数
150  NEXT I          ┘
```

☆機械語プログラムの呼び出し方☆

機械語で書いたプログラムをＢＡＳＩＣのプログラムで利用するには、ＵＳＲ関数（ユーザー関数）として呼び出す方法と、ＣＡＬＬ文でサブルーチンとして呼び出す方法の二通りある。

《ＵＳＲ関数》

ＵＳＲ関数で機械語のプログラムを呼び出すと、ＢＡＳＩＣの関数と同じように使える。計算の結果はＵＳＲ関数自体が持つので、計算式の中でＵＳＲ関数を使ったり、ＩＦ文の条件に書くなど、さまざまに使える。ＵＳＲ関数に返す値の型は、引き数の型と同じにする。引き数には定数、変数のほかに式が書ける。ただし、引き数は一つしか書けない。ＵＳＲ関数を呼び出すには、ＤＥＦ　ＵＳＲ文で機械語のプログラムの先頭番地を宣言しておく必要がある。

（例）
```
10   CLEAR ,&HE000       機械語エリアをE000H番地から確保する
20   BLOAD "PROG0"       機械語プログラムをロードする

100  DEF USR0=&HE000     USR関数の先頭番地を宣言する
110  INPUT A
120  B=USR0 (A)          USR関数を呼び出す
130  PRINT A,B
140  END
```

《ＣＡＬＬ文》

ＣＡＬＬ文で機械語プログラム（機械語サブルーチン）を呼び出して使うときは、ＵＳＲ関数とは異なり複数の引き数を受け渡しできる。引き数は機械語サブルーチンに値を渡すだけでなく、値を返すためにも使う。ただし、引き数には変数しか書けない。機械語サブルーチンを呼び出すには、先頭番地をサブルーチン名（変数）にセットしておく必要がある。

（例）
```
10   CLEAR ,&HE000       機械語エリアをE000H番地から確保する
20   BLOAD "PROG1"       機械語プログラムをロードする

100  SUB1=&HE100         機械語サブルーチン名に先頭番地を入れる
110  INPUT A
120  CALL SUB1 (A,B)     機械語サブルーチンを呼び出す
130  PRINT A,B
140  END
```

☆機械語プログラムの作り方☆

《機械語プログラムの構造》

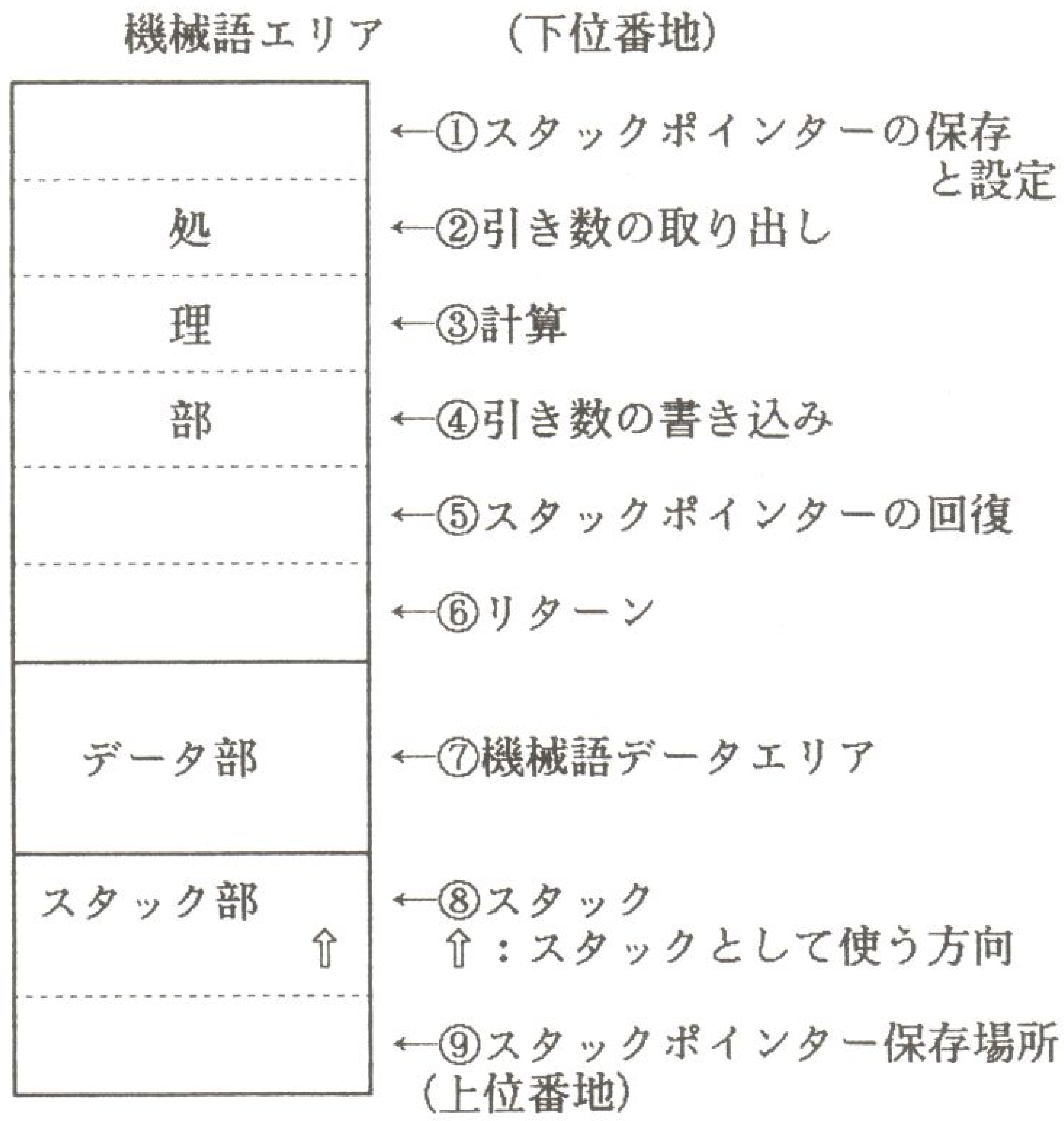
機械語プログラムは、右図の①～⑥の処理部と⑦～⑨のデータ部とスタック部から成る。

機械語プログラムがＵＳＲ関数やＣＡＬＬ文から呼ばれたとき、『入り口の処理』でスタックポインターを保存してから設定する。『入り口の処理』はＵＳＲ関数とＣＡＬＬ文で共通。また、ＵＳＲ関数やＣＡＬＬ文に戻るとき、スタックポインターを元に戻す『出口の処理』も共通である。

引き数を取り出したり書き込んだりするとき、ＵＳＲ関数とＣＡＬＬ文では引き数の扱い方がちがう。また、引き数が数値か文字列かによって、ＵＳＲ関数とＣＡＬＬ文それぞれで扱い方が異なる。

《入り口と出口の処理》

機械語プログラムからＢＡＳＩＣに戻るには必ずＲＥＴ命令を使う。ＢＡＳＩＣは機械語プログラムを機械語のＣＡＬＬ命令で呼び出しているため、スタックには戻り番地として２バイト入っている。ＢＡＳＩＣに戻るときには必ずスタックポインターは呼び出されたときと同じ番地を指すようにする。



《スタックの処理》

スタックを８レベル（１６バイト）を超えて使うときには、機械語エリアにスタック部を用意する。スタック部の最後の２バイトをスタックポインターの保存場所として、それ以前をスタックとして使う。スタックが８レベル以下のときは、スタック部を用意しなくてよい。スタックポインターの処理は次のようにする。

・スタックポインターの保存と設定
（ｎｎｎｎ番地は⑨スタックポインター保存場所）

```
  スタックポインターの値をｎｎｎｎ番地に保存する
        ＬＸＩ　　Ｈ，０            ＨＬに０をセットする
        ＤＡＤ　　ＳＰ              ＳＰの値をＨＬにセットする
        ＳＨＬＤ　ｎｎｎｎ          ＨＬの値をｎｎｎｎとｎｎｎｎ＋１番地に保存する
  スタックポインターを機械語エリアのスタックに設定する
        ＬＸＩ　　ＳＰ，ｎｎｎｎ    ｎｎｎｎ番地をスタックポインターに設定する
```

・スタックポインターの回復

```
        ＬＨＬＤ　ｎｎｎｎ          ｎｎｎｎとｎｎｎｎ＋１番地の値をＨＬにセットする
        ＳＰＨＬ                    ＨＬをＳＰにセットする
```

☆引き数の取り出しと書き込み☆

《ＵＳＲ関数を使う場合の引き数の扱い》

（１）引き数が数値の場合

機械語プログラムがＵＳＲ関数から呼ばれたときは、引き数の型がＡレジスターに入り、引き数がＦＡＣ（浮動小数点アキュムレーター）に入る。ＦＡＣはＢＡＳＩＣが使う８バイトの作業場所で、引き数が整数型か単精度実数型か倍精度実数型かによって、各バイトの働きが異なる。ＨＬレジスターにはＦＡＣの先頭から５バイト目の番地が渡される。

機械語プログラムでは、Ａレジスターに引き数の型を、ＦＡＣに計算結果をセットして、ＵＳＲ関数に返す。

ＵＳＲ関数と機械語プログラムで引き数をやりとりするときのＡレジスター、ＨＬレジスター、ＦＡＣの働きはつぎの表の通り。

引き数の型	整数型	単精度実数型	倍精度実数型
ＨＬレジスターの値	ＦＡＣの５バイト目の番地		
Ａレジスターの値	2	4	8
ＦＡＣの１バイト目			仮数の最下位８ｂｉｔ
ＦＡＣの２バイト目			仮数の第６位８ｂｉｔ
ＦＡＣの３バイト目			仮数の第５位８ｂｉｔ
ＦＡＣの４バイト目			仮数の第４位８ｂｉｔ
ＦＡＣの５バイト目	下位８ｂｉｔ	仮数の下位８ｂｉｔ	仮数の第３位８ｂｉｔ
ＦＡＣの６バイト目	上位８ｂｉｔ	仮数の中位８ｂｉｔ	仮数の第２位８ｂｉｔ
ＦＡＣの７バイト目		仮数の上位８ｂｉｔ	仮数の最上位８ｂｉｔ
ＦＡＣの８バイト目		指数部	指数部

（例）引き数が整数の場合

引き数が整数のとき、ＵＳＲ関数から呼ばれる機械語プログラムは次のように作る。

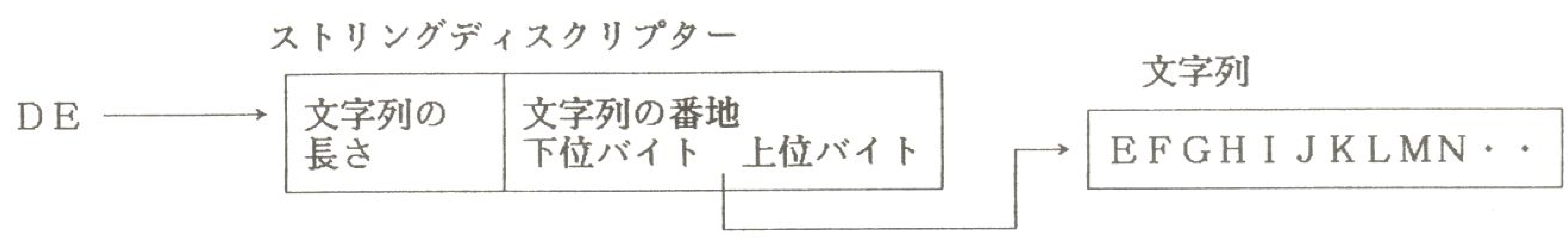
```
ＦＡＣの場所を保存する
        ＳＨＬＤ　ｎｎｎｎ        ＨＬに入ってくるＦＡＣの場所を保存する
数値をＦＡＣからＨＬに取り出す
        ＭＯＶ　　Ｅ，Ｍ          下位バイトをＥに取り出す
        ＩＮＸ　　Ｈ              ＨＬを次の番地にセットする
        ＭＯＶ　　Ｄ，Ｍ          上位バイトをＤに取り出す
        ＸＣＨＧ                  ＤＥに入れた値をＨＬに入れる

    ＨＬに取り出した数値を使い計算する。結果をＨＬに入れる

ＨＬに入っている計算結果をＦＡＣに入れる
        ＸＣＨＧ                  ＨＬにある計算結果をＤＥに移す
        ＬＨＬＤ　ｎｎｎｎ        ＨＬにＦＡＣの場所をセットする
        ＭＯＶ　　Ｍ，Ｅ          下位バイトをＦＡＣに入れる
        ＩＮＸ　　Ｈ              ＨＬを次の番地にセットする
        ＭＯＶ　　Ｍ，Ｄ          上位バイトをＦＡＣに入れる
ＢＡＳＩＣプログラムに戻る
        ＲＥＴ
```

（２）引き数が文字列の場合

機械語プログラムがＵＳＲ関数から呼ばれたときは、引き数の型（＝３）がＡレジスターに入り、ストリングディスクリプターの番地がＤＥレジスターに入って渡される。ストリングディスクリプターは３バイトで、引き数の文字列の長さと番地を示す。機械語プログラムの中でストリングディスクリプターの内容を変えてはいけない。



（例）文字列を取り出すには、次のようにする

文字列の場所（番地）をＤＥに、文字列の長さをＣに入れる

```
XCHG            HLにストリングディスクリプターの場所を取り出す
MOV   C,M       Cにストリングの長さを取り出す
INX   H         HLを次の番地にセットする
MOV   E,M       Eに下位バイトを取り出す
INX   H         HLを次の番地にセットする
MOV   D,M       Dに上位バイトを取り出す
```

文字列の先頭１文字をＡに取り出す

```
XCHG            HLに文字列の場所を移す
MOV   A,M       Aに文字列の先頭の１文字を取り出す
```

文字列を置き換えるには、次のようにする

```
MOV   M,A       文字列の先頭にAの内容を入れる
```

《ＣＡＬＬ文を使うときの引き数の扱い》

機械語プログラムがＣＡＬＬ文から呼ばれたときは、引き数の番地が渡される。

引き数の番地の表し方は、数値か文字列かで異なる。数値の場合は、整数型（２バイト）、単精度実数型（４バイト）、倍精度実数型（８バイト）のそれぞれの引き数の先頭番地を指す。文字列の場合はストリングディスクリプターの先頭番地を指す。

機械語プログラムからＣＡＬＬ文に結果を返すときは、直接引き数に値を入れる。

ＣＡＬＬ文の引き数の個数によって、引き数の受け渡し方がちがう。

・３個以下の引き数の扱い

引き数の個数が３個以下のとき、各引き数の番地はレジスターＨＬ・ＤＥ・ＢＣに入っている。

ＣＡＬＬ　機械語プログラム（引き数１，引き数２，引き数３）

ＨＬ：引き数１の番地←
ＤＥ：引き数２の番地←
ＢＣ：引き数３の番地←

・４個以上の引き数の扱い

引き数の個数が４個以上のとき、引き数１の番地と引き数２の番地がレジスターＨＬとＤＥに入る。３個目からあとの引き数の番地は引き数テーブルに入り、ＢＣレジスターが引き数テーブルの場所を示す。

ＣＡＬＬ　機械語プログラム（引き数１，引き数２，引き数３，引き数４，引き数５，・・・）

ＨＬ：引き数１の番地←
ＤＥ：引き数２の番地←
ＢＣ：引き数テーブルの番地 →

引き数テーブル

引き数３の番地	引き数４の番地	～

（例）引き数が数値の場合

引き数が４個以上のときは、引き数の取り出しに引き数テーブルを使う。
また、３番目以降の引き数が数値の場合は、次のようにして引き数を取り出す。

ＣＡＬＬ　ＳＵＢ（Ａ％，Ｂ％，Ｃ％，Ｄ％）の場合

ＨＬ：１番目の引き数Ａ％の番地
ＤＥ：２番目の引き数Ｂ％の番地
ＢＣ：引き数テーブルの番地 →

引き数テーブル

３番目の引き数Ｃ％の番地	４番目の引き数Ｄ％の番地

→ ４番目の引き数

Ａ％の番地はＨＬレジスターに、Ｂ％の番地はＤＥレジスターに入っているので、各レジスターを使って取り出す。Ｃ％とＤ％の番地はＢＣレジスターの指す引き数テーブルにある。

４番目の引き数Ｄ％の値をＨＬレジスターに取り出すには次のようにする。

引き数テーブルからＤ％の番地をＤＥレジスターに取り出す

```
        ＰＵＳＨ  Ｂ          ＢＣにある引き数テーブルの番地をＨＬに取り出す
        ＰＯＰ    Ｈ
        ＩＮＸ    Ｈ          ＨＬを変数Ｄ％の番地にセットする。引き数テーブルの
        ＩＮＸ    Ｈ          ２番目だから２増やす
        ＭＯＶ    Ｅ，Ｍ      Ｄ％の番地の下位バイトを取り出す
        ＩＮＸ    Ｈ          ＨＬを次の番地にセットする
        ＭＯＶ    Ｄ，Ｍ      Ｄ％の番地の上位バイトを取り出す
```

引き数Ｄ％の値をＨＬレジスターに取り出す

```
        ＬＤＡＸ  Ｄ          Ｄ％の値の下位バイトを取り出す
        ＭＯＶ    Ｌ，Ａ
        ＩＮＸ    Ｄ          ＤＥを次の番地にセットする
        ＬＤＡＸ  Ｄ          Ｄ％の値の上位バイトを取り出す
        ＭＯＶ    Ｈ，Ａ
```

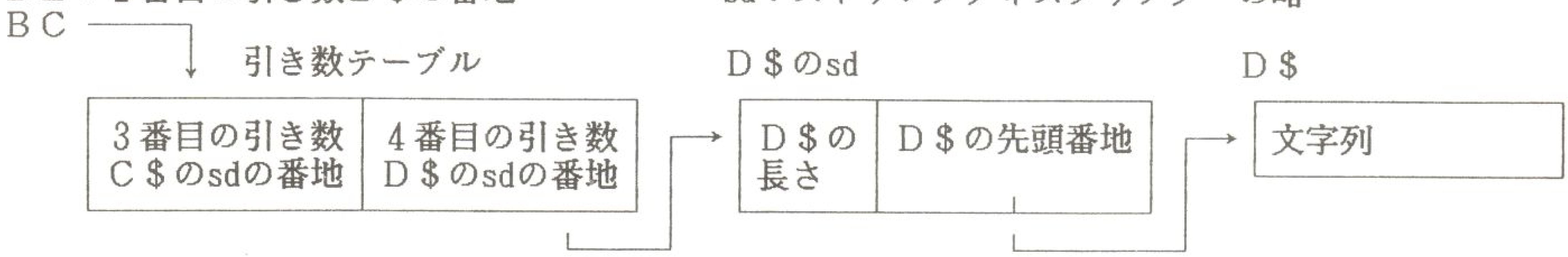
ＨＬレジスターの内容をＤ％に入れるには次のようにする
（ＤＥレジスターにＤ％の番地が入っているものとする）

```
        ＭＯＶ    Ａ，Ｌ      下位バイトをＤ％にしまう
        ＳＴＡＸ  Ｄ
        ＩＮＸ    Ｄ          ＤＥを次の番地にセットする
        ＭＯＶ    Ａ，Ｈ      上位バイトをＤ％にしまう
        ＳＴＡＸ  Ｄ
```

（例）引き数が文字列の場合
３番目以降の引き数が文字列のとき、文字列を取り出すには次のようにする。

ＣＡＬＬ　ＳＵＢ（Ａ＄，Ｂ＄，Ｃ＄，Ｄ＄）の場合

ＨＬ：１番目の引き数Ａ＄の番地
ＤＥ：２番目の引き数Ｂ＄の番地　　　　sd：ストリングディスクリプターの略



Ａ＄とＢ＄のストリングディスクリプターの番地はＨＬレジスターとＤＥレジスターに入っている。
Ｃ＄とＤ＄のストリングディスクリプターの番地は引き数テーブルにあり、そのテーブルをＢＣレジスターが指している。

Ｄ＄の最初の１文字をＡに取り出すには次のようにする

```
        ＰＵＳＨ  Ｂ          ＢＣにある引き数テーブルの番地をＨＬに取り出す
        ＰＯＰ    Ｈ
        ＩＮＸ    Ｈ          ＨＬをＤ％の番地にセットする。引き数テーブルの２番
        ＩＮＸ    Ｈ          目だから２増やす
        ＭＯＶ    Ｅ，Ｍ      Ｄ＄のストリングディスクリプターの番地をＤＥに取り
        ＩＮＸ    Ｈ          出す
        ＭＯＶ    Ｄ，Ｍ
        ＸＣＨＧ              取り出した番地をＨＬに入れる
        ＭＯＶ    Ｃ，Ｍ      Ｄ＄の長さをＣに取り出す
        ＩＮＸ    Ｈ          ＨＬを次の番地にセットする
        ＭＯＶ    Ｅ，Ｍ      Ｄ＄の番地をＤＥに取り出す
        ＩＮＸ    Ｈ
        ＭＯＶ    Ｄ，Ｍ
        ＬＤＡＸ  Ｄ          Ｄ＄の１文字目をＡに取り出す
```

文字列を置き換えるには次のようにする

```
        ＳＴＡＸ  Ｄ          文字列の先頭にＡの内容を入れる
```

11．8801　機械語プログラム集

機械語プログラムはモニターまたはＰＯＫＥ命令でメモリー上に作るか、すでにセーブしてあるものを、ＢＬＯＡＤ命令でロードする。詳しくは「ＵＳＲ関数とＣＡＬＬ文の使い方」参照。

☆（８８）機械語　例題１☆

１．機能

整数型引き数をビットパターンとして扱い、値が１のビット数を数える。

２．呼び出し形式

ＤＥＦ　ＵＳＲ１＝＆ＨＥ０００
ＰＲＩＮＴ　ＵＳＲ１（整数型引き数）

３．利用例

－３から３までの整数の１になっているビット数を数え、ビット数が奇数か偶数かを表示する。

```
100 DEF USR1=&HE000
110 DIM P$(1): P$(0)="ｸﾞｳｽｳ": P$(1)="ｷｽｳ"
120 FOR I%=-3 TO 3
130   PRINT USING "### ## &   &";I%,USR1(I%),P$(USR1(I%) MOD 2)
140 NEXT I%
run
 -3 15 ｷｽｳ
 -2 15 ｷｽｳ
 -1 16 ｸﾞｳｽｳ
  0  0 ｸﾞｳｽｳ
  1  1 ｷｽｳ
  2  1 ｷｽｳ
  3  2 ｸﾞｳｽｳ
```

４．制約事項

整数型以外の引き数ではなにもしない。このときＵＳＲ１の値は引き数と同じになる。

５．機械語プログラム

入り口状態　：Ａ　－　引き数の型番号（整数型ならば２）
　　　　　　　ＨＬ－　ＦＡＣの整数型引き数の下位バイトの番地

```
                                引き数の型をチェックする
E000 FE 02     CPI   02            整数型（型番号が２）かどうか調べる
E002 C0        RNZ                 整数型でなければ戻る
                                引き数の値を取り出す
E003 5E        MOV   E,M           Ｅに引き数の値の下位バイトを取り出す
E004 23        INX   H             ＨＬをＦＡＣの引き数の上位バイトの番地にする
E005 56        MOV   D,M           Ｄに引き数の値の上位バイトを取り出す
E006 EB        XCHG                ＨＬに引き数の値、ＤＥにＦＡＣの上位バイトの番地を移す
                                繰り返し動作の準備をする
E007 06 10     MVI   B,10          Ｂに繰り返し数１６（ビットの数）を設定する
E009 0E 00     MVI   C,00          １になっているビット数のカウンターＣをクリアーする
                                ＨＬの最上位ビット（ＭＳＢ）を調べ、１ならばカウントする
E00B 29        DAD   H             ＨＬを１ビット左にずらしＭＳＢをＣフラグに入れる
E00C 30 01     JRNC  E00F          Ｃフラグがゼロならカウントする部分をとばす
E00E 0C        INR   C             １になっているビット数のカウンターＣに１をたす
                                繰り返し動作を指示する
E00F 10 FA     DJNZ  E00B          未処理ビット数Ｂから１を引きゼロでなければ繰り返す
                                ＵＳＲ関数の返す値（関数値）を設定する
E011 EB        XCHG                ＨＬにＦＡＣの上位バイトの番地を移す
E012 36 00     MVI   M,00          関数値の上位バイトをクリアーする
E014 2B        DCX   H             ＨＬをＦＡＣの関数値の下位バイトの番地にする
E015 71        MOV   M,C           関数値の下位バイトに１であるビット数Ｃをしまう
E016 C9        RET              呼び出しプログラムに戻る
```

☆（８８）機械語　例題２☆

１．機能

文字列型の引き数をもらい、その文字列の中でプリンターに印字できない文字（文字コード００Ｈ～１ＦＨ・７ＦＨ・Ｆ８Ｈ～ＦＦＨ）を「・」（文字コードＡ５Ｈ）に置き換える。

２．呼び出し形式

ＤＥＦ　ＵＳＲ２＝&ＨＥ０３０
ＰＲＩＮＴ　ＵＳＲ２（文字列型引き数）

３．利用例

文字コード順に並んだ文字の表を印字する。

```
100 DEF USR2=&HE030
110 FOR J=0 TO &HF
120  S$=""
130  FOR I=0 TO &HF0 STEP &H10
140   S$=S$+CHR$(I+J)+" "
150  NEXT I
160  LPRINT USR2(S$)
170 NEXT J
run
· ·   0 @ P ` p ▁ ┴   ー タ ミ ═ ╳
· · ! 1 A Q a q ▂ ┬ 。 ア チ ム ╞ 円
· · " 2 B R b r ▃ ┤ 「 イ ツ メ ╪ 年
· · # 3 C S c s ▄ ├ 」 ウ テ モ ╡ 月
· · $ 4 D T d t ▅ ￣ 、 エ ト ヤ ◢ 日
· · % 5 E U e u ▆ ─ ・ オ ナ ユ ◣ 時
· · & 6 F V f v ▇ │ ヲ カ ニ ヨ ◥ 分
· · ' 7 G W g w █ ▕ ァ キ ヌ ラ ◤ 秒
· · ( 8 H X h x ▏ ┌ ィ ク ネ リ ♠ ·
· · ) 9 I Y i y ▎ ┐ ゥ ケ ノ ル ♥ ·
· · * : J Z j z ▍ └ ェ コ ハ レ ♦ ·
· · + ; K [ k { ▌ ┘ ォ サ ヒ ロ ♣ ·
· · , < L ¥ l ¦ ▋ ╭ ャ シ フ ワ ● ·
· · - = M ] m } ▊ ╮ ュ ス ヘ ン ○ ·
· · . > N ^ n ~ ▉ ╰ ョ セ ホ ゛ ╱ ·
· · / ? O _ o · ┼ ╯ ッ ソ マ ゜ ╲ ·
```

４．制約事項

文字列型以外の引き数ではなにもしない。このときＵＳＲ２の値は引き数と同じになる。

５．機械語プログラム

入り口状態　：Ａ　－　引き数の型番号（文字列型ならば３）
　　　　　　　ＤＥ－　文字列型引き数のストリングディスクリプター（ｓｄ）の先頭番地

```
                              引き数の型をチェックする
E030 FE 03      CPI  03          文字列型（型番号が３）かどうか調べる
E032 C0         RNZ              文字列型でなければ戻る
                              引き数文字列の長さをチェックする
E033 EB         XCHG             ＨＬに引き数の文字列のｓｄの先頭番地を移す
E034 7E         MOV  A,M         Ａに文字列の長さを取り出す
E035 B7         ORA  A           文字列の長さＡがゼロかどうか調べる
E036 C8         RZ               文字列の長さＡがゼロならば戻る
E037 47         MOV  B,A         Ｂに文字列の長さを移す（繰り返しの回数になる）
                              引き数の文字列の先頭番地を取り出す
E038 23         INX  H           ＨＬをｓｄの２バイト目の番地にする
E039 5E         MOV  E,M         Ｅに引き数の文字列の先頭番地の下位バイトを取り出す
E03A 23         INX  H           ＨＬをｓｄの３バイト目の番地にする
E03B 56         MOV  D,M         Ｄに文字列の先頭番地の上位バイトを取り出す
E03C EB         XCHG             ＨＬに上で取り出した引き数の文字列の先頭番地を移す
```

```
                                HLの指している文字を調べ、置き換えが必要かどうか決める
E03D 7E        MOV  A,M           HLの指している文字をAに取り出す
E03E FE 20     CPI  20            文字コード20Hと比べる
E040 38 0C     JRC  E04E          20H未満ならば置き換え処理へ行く
E042 FE 7F     CPI  7F            文字コード7FHと比べる
E044 38 0A     JRC  E050          7FH未満ならば置き換え処理をとばす
E046 FE 80     CPI  80            文字コード80Hと比べる
E048 38 04     JRC  E04E          80H未満ならば置き換え処理へ行く
E04A FE F8     CPI  F8            文字コードF8Hと比べる
E04C 38 02     JRC  E050          F8H未満ならば置き換え処理をとばす
                                置き換え処理をする
E04E 36 A5     MVI  M,A5          HLの指している文字を「・」(文字コードA5H)にする
                                繰り返し動作を指示する
E050 23        INX  H             HLを引き数の文字列の次の文字の番地にする
E051 10 EA     DJNZ E03D          残りの文字数Bから1を引きゼロでなければ繰り返す
E053 C9        RET              呼び出しプログラムに戻る
```

☆(88)機械語　例題3☆

1．機能

１６進数の数字（０～９・Ａ～Ｆまたはａ～ｆ）から成る文字列を内部形（２進数表現）に変換する。１６進数の数字２文字（２バイト）が内部形１バイトに相当する。

2．呼び出し形式

ＨＥＸＶＡＬ＝＆ＨＥ０６０
ＣＡＬＬ　ＨＥＸＶＡＬ（文字列変数１，文字列変数２）

文字列変数１（入力）　：１６進数の数字から成る文字列
文字列変数２（出力）　：変換した値を入れる文字列（文字列変数１の半分の長さであること）

3．利用例

プリンターに漢字かな混じりの文を印字する。

```
100 HEXVAL=&HE060
110 H$="1b4b4c5c1b481b26c6d4bbbcb21b4b442b467c3f374a391b481b24"
120 V$=SPACE$(LEN(H$)/2)
130 CALL HEXVAL(H$,V$)
140 LPRINT V$
run
目にやさしい朝日新聞
```

4．制約事項

・引き数の数や型が呼び出し形式と違う場合、正しく動かない。
・文字列変数２の長さが文字列変数１の長さの半分でない場合は何もしない。
・文字列変数１の中に１６進数の数字以外の文字がある場合には、変換した結果は正しくない。

5．機械語プログラム

入り口状態　：ＨＬ－　文字列変数１のストリングディスクリプター（ｓｄ１）の先頭番地
　　　　　　　ＤＥ－　文字列変数２のストリングディスクリプター（ｓｄ２）の先頭番地

```
                                文字列変数の長さをチェックする
E060 1A        LDAX D             Aに文字列変数2の長さを取り出す
E061 B7        ORA  A             文字列変数2の長さAがゼロかどうか調べる
E062 C8        RZ                 文字列変数2の長さAがゼロならば戻る
E063 47        MOV  B,A           Bに文字列変数2の長さを移す(繰り返しの回数になる)
E064 7E        MOV  A,M           Aに文字列変数1の長さを取り出す
E065 0F        RRC                長さを2で割り余りをCフラグに入れる
E066 D8        RC                 余りがあれば(長さが奇数ならば)戻る
E067 B8        CMP  B             文字列変数1の長さの半分と文字列変数2の長さBを比べる
```

```
E068 C0         RNZ               同じでなければ戻る
                                文字列変数の先頭番地を取り出す
E069 23         INX  H            ＨＬをｓｄ１の２バイト目の番地にする
E06A CD E094    CALL E094         ＨＬに文字列変数１の文字列の先頭番地を取り出す
E06D EB         XCHG              ＨＬにｓｄ２の番地、ＤＥに文字列変数１の先頭番地を得る
E06E 23         INX  H            ＨＬをｓｄ２の２バイト目の番地にする
E06F CD E094    CALL E094         ＨＬに文字列変数２の文字列の先頭番地を取り出す
                                文字列変数１の２文字を変換して文字列変数２へしまう
E072 1A         LDAX D            Ａに文字列変数１の１文字を取り出す
E073 CD E087    CALL E087         Ａにある１６進数の数字を内部形に変換し結果をＡに入れる
E076 87         ADD  A            左に１ビットずらす
E077 87         ADD  A            左に１ビットずらす
E078 87         ADD  A            左に１ビットずらす
E079 87         ADD  A            左に１ビットずらす（Ａの下４ビットを上４ビットに移す）
E07A 4F         MOV  C,A          Ｃに４ビットずらした結果を入れる
E07B 13         INX  D            ＤＥを文字列変数１の次の文字の番地にする
E07C 1A         LDAX D            Ａに文字列変数１の次の１文字を取り出す
E07D CD E087    CALL E087         Ａにある１６進数の数字を内部形に変換し結果をＡに入れる
E080 B1         ORA  C            前に求めた上４ビットの値Ｃと合成する
E081 77         MOV  M,A          合成した値を引き数の文字列２にしまう
                                繰り返し動作を指示する
E082 13         INX  D            ＤＥを文字列変数１の次の文字の番地にする
E083 23         INX  H            ＨＬを文字列変数２の次のバイトの番地にする
E084 10 EC      DJNZ E072         未処理バイト数Ｂから１を引きゼロでなければ繰り返す
E086 C9         RET             呼び出しプログラムに戻る

                    内部サブルーチン
                      機能       ：Ａにある１６進数の数字を内部形に変換し結果をＡに入れる
                                   Ａの内容が１６進数の数字でないとき変換結果は意味がない
                      入り口状態 ：Ａ－１６進数の数字（文字コード）
                      戻り状態   ：Ａ－変換結果（内部形）
                                   Ａ・Ｆレジスター以外のレジスターは変化しない
                      使用スタック：  なし

E087 D6 30      SUI  30           文字「０」～「９」が内部形０～９になるようにする
E089 FE 0A      CPI  0A           結果が１０未満かどうか調べる
E08B D8         RC                結果が１０未満ならば呼び出し機械語プログラムに戻る
E08C D6 07      SUI  07           文字「Ａ」～「Ｆ」が内部形１０～１５になるようにする
E08E FE 10      CPI  10           結果が１６未満かどうか調べる
E090 D8         RC                １６未満ならば呼び出し機械語プログラムに戻る
E091 D6 20      SUI  20           文字「ａ」～「ｆ」が内部形１０～１５になるようにする
E093 C9         RET               呼び出し機械語プログラムに戻る

                    内部サブルーチン
                      機能       ：ＨＬの指している番地から２バイトをＨＬに取り出す
                      入り口状態 ：ＨＬ－値を取り出す先頭番地
                      戻り状態   ：ＨＬ－取り出した値
                                   ＨＬ以外のレジスターは変化しない
                      使用スタック：  １レベル

E094 D5         PUSH D            作業用レジスターＤＥの内容をスタックに保存する
E095 5E         MOV  E,M          ＨＬの指している番地の内容をＥに取り出す（下位バイト）
E096 23         INX  H            ＨＬを次のバイトの番地にする
E097 56         MOV  D,M          ＨＬの指している番地の内容をＤに取り出す（上位バイト）
E098 EB         XCHG              ＨＬに取り出した値をもってくる
E099 D1         POP  D            スタックから保存したＤＥの内容を取り戻す
E09A C9         RET               呼び出し機械語プログラムに戻る
```

☆（８８）機械語　例題４☆

１．機能

文字列を内部形２進数として受け取り、１６進数の数字（０～９・Ａ～Ｆ）に変換する。内部形１バイトが１６進数の数字２文字（２バイト）に相当する。

２．呼び出し形式

ＶＡＬＨＥＸ＝＆ＨＥ０Ａ０
ＣＡＬＬ　ＶＡＬＨＥＸ（文字列変数１，文字列変数２）
　　文字列変数１（入力）　：内部形２進数として扱われる文字列
　　文字列変数２（出力）　：変換した１６進数の数字を入れる文字列（長さは文字列変数１の長さの２倍であること）

３．利用例

整数型・単精度実数型・倍精度実数型の数値の内部形式を表示する。

```
100 VALHEX=&HE0A0
110 FOR I=-3 TO 3
120   VI$=MKI$(I): HI$=SPACE$( 4): CALL VALHEX(VI$,HI$)
130   VS$=MKS$(I): HS$=SPACE$( 8): CALL VALHEX(VS$,HS$)
140   VD$=MKD$(I): HD$=SPACE$(16): CALL VALHEX(VD$,HD$)
150   PRINT I,HI$,HS$,HD$
160 NEXT I
run
-3           FDFF          0000C082          000000000000C082
-2           FEFF          00008082          0000000000008082
-1           FFFF          00008081          0000000000008081
 0           0000          00008000          0000000000008000
 1           0100          00000081          0000000000000081
 2           0200          00000082          0000000000000082
 3           0300          00004082          0000000000004082
```

４．制約事項

・引き数の数や型が呼び出し形式と違う場合、正しく動かない。
・文字列変数２の長さが文字列変数１の長さの２倍でない場合はなにもしない。

５．機械語プログラム

入り口状態：ＨＬ－　文字列変数１のストリングディスクリプター（ｓｄ１）の先頭番地
　　　　　　ＤＥ－　文字列変数２のストリングディスクリプター（ｓｄ２）の先頭番地

```
                                 文字列変数の長さをチェックする
E0A0 7E         MOV  A,M            Aに文字列変数１の長さを取り出す
E0A1 B7         ORA  A              文字列変数１の長さAがゼロかどうか調べる
E0A2 C8         RZ                  文字列変数１の長さAがゼロならば戻る
E0A3 47         MOV  B,A            Bに文字列変数１の長さを移す（繰り返しの回数になる）
E0A4 1A         LDAX D              Aに文字列変数２の長さを取り出す
E0A5 0F         RRC                 長さを２で割り、余りをCフラグに入れる
E0A6 D8         RC                  余りがあれば（長さが奇数ならば）戻る
E0A7 B8         CMP  B              文字列変数２の長さの半分Bと文字列変数１の長さを比べる
E0A8 C0         RNZ                 同じでなければ戻る
                                 文字列変数の文字列の先頭番地を取り出す
E0A9 23         INX  H              HLをsd1の２バイト目の番地にする
E0AA CD E0D0    CALL E0D0           HLに文字列変数１の文字列の先頭番地を取り出す
E0AD EB         XCHG                HLにsd2の番地、DEに文字列変数１の先頭番地を移す
E0AE 23         INX  H              HLをsd2の２バイト目の番地にする
E0AF CD E0D0    CALL E0D0           HLに文字列変数２の文字列の先頭番地を取り出す
                                 文字列変数１の１バイトを変換して文字列変数２へしまう
E0B2 1A         LDAX D              Aに文字列変数１の１バイトを取り出す
E0B3 0F         RRC                 右に１ビットずらす
E0B4 0F         RRC                 右に１ビットずらす
E0B5 0F         RRC                 右に１ビットずらす
E0B6 0F         RRC                 右に１ビットずらす（Aの上４ビットを下４ビットに移す）
```

```
E0B7 CD E0C6   CALL E0C6        Aの下4ビットを16進数の数字1文字に変換する
E0BA 77        MOV  M,A         変換した16進数の数字を文字列変数2へしまう
E0BB 23        INX  H           HLを文字列変数2の次の文字の番地にする
E0BC 1A        LDAX D           Aに文字列変数1の1バイトを再び取り出す
E0BD CD E0C6   CALL E0C6        Aの下4ビットを16進数の数字1文字に変換する
E0C0 77        MOV  M,A         変換した16進数の数字を文字列変数2へしまう
                              繰り返し動作を指示する
E0C1 13        INX  D           DEを文字列変数1の次のバイトの番地にする
E0C2 23        INX  H           HLを文字列変数2の次の文字の番地にする
E0C3 10 ED     DJNZ E0B2        残りバイト数Bから1を引きゼロでなければ繰り返す
E0C5 C9        RET            呼び出しプログラムに戻る
```

内部サブルーチン

機能　　　　：Aの下4ビットにある2進数を16進数の数字に変換しAに返す
入り口状態　：A－変換する値（下4ビットが有効・上4ビットは無視）
戻り状態　　：A－変換した結果の16進数の数字
　　　　　　　A・Fレジスター以外のレジスターは変化しない
使用スタック：　なし

```
E0C6 E6 0F     ANI  0F          Aの下4ビットを取り出し上4ビットは捨てる
E0C8 C6 30     ADI  30          内部形0～9が文字「0」～「9」になるようにする
E0CA FE 3A     CPI  3A          「9」以下（文字コード3AH未満）かどうか調べる
E0CC D8        RC               「9」以下ならば呼び出し機械語プログラムに戻る
E0CD C6 07     ADI  07          内部形10～15が文字「A」～「F」になるようにする
E0CF C9        RET              呼び出し機械語プログラムに戻る
```

内部サブルーチン

機能　　　　：HLの指している番地から2バイトをHLに取り出す
入り口状態　：HL－値を取り出す先頭番地
戻り状態　　：HL－取り出した値
　　　　　　　HL以外のレジスターは変化しない
使用スタック：　1レベル

```
E0D0 D5        PUSH D           作業用レジスターDEの内容をスタックに保存する
E0D1 5E        MOV  E,M         HLの指している番地の内容をEに取り出す（下位バイト）
E0D2 23        INX  H           HLを次のバイトの番地にする
E0D3 56        MOV  D,M         HLの指している番地の内容をDに取り出す（上位バイト）
E0D4 EB        XCHG             HLに取り出した値をもってくる
E0D5 D1        POP  D           スタックから保存したDEの内容を取り戻す
E0D6 C9        RET              呼び出し機械語プログラムに戻る
```

☆（88）機械語　例題5☆

1．機能

指定した番地からメモリーの内容を文字列に取り出す。取り出す長さは文字列の長さとする。

2．呼び出し形式

ＰＥＥＫＬ＝＆ＨＥ０Ｅ０
ＣＡＬＬ　ＰＥＥＫＬ（整数変数，文字列変数）
　整数変数（入力）　　：メモリーの内容を取り出す開始番地
　文字列変数（出力）　：取り出したメモリーの内容を入れる文字列（長さが取り出す長さ）

3．利用例

指定したメモリーの内容を16進数と文字形式でプリンターに印字する。
機械語例題2（ＵＳＲ2）・例題4（ＶＡＬＨＥＸ）も利用する。

```
100 DEF USR2=&HE030: VALHEX=&HE0A0: PEEKL=&HE0E0
110 INPUT "ｶｲｼ ﾊﾞﾝﾁ";S
120 INPUT "ｼｭｳﾘｮｳ ﾊﾞﾝﾁ";E
130 FOR I%=S TO E STEP &H10
```

```
140  IF E-I%+1>&H10 THEN L=&H10 ELSE L=E-I%+1
150  V$=SPACE$(L): CALL PEEKL(I%,V$)
160  H$=SPACE$(L*2): CALL VALHEX(V$,H$)
170  LPRINT HEX$(I%);TAB(6);USR2(V$);TAB(24);H$
180 NEXT I%
run
ｶｲｼ ﾊﾞﾝﾁ? &h4e40
ｼｭｳﾘｮｳ ﾊﾞﾝﾁ? &h4e85
4E40  ・−N・ｷ・年CNH■・ｷ ﾆ   FA2D4E1AB713F2434E48E1E51AB720C6
4E50  ﾃ MKYBD・COM1・COM   C3A04D4B594244FF434F4D31FE434F4D
4E60  2・COM3・CAS1・CAS2   32FD434F4D33FC43415331FB43415332
4E70  ・LPT1・SCRN・・_z・z   FA4C505431F95343524EF800807A127A
4E80  (z(z>z             287A287A3E7A
```

4．制約事項

　引き数の数や型が呼び出し形式と違う場合、正しく動かない。

5．機械語プログラム

入り口状態：ＨＬ－　整数変数の先頭番地

ＤＥ－　文字列変数のストリングディスクリプター（ｓｄ）の先頭番地

```
                          引き数の情報を取り出す
E0E0 CD E0F3  CALL E0F3     ＨＬに整数変数の値を取り出す
E0E3 EB       XCHG          ＨＬに文字列の先頭番地、ＤＥに整数変数の値を移す
E0E4 7E       MOV  A,M      Ａに文字列変数の長さを取り出す
E0E5 B7       ORA  A        文字列変数の長さＡがゼロかどうか調べる
E0E6 C8       RZ            文字列変数の長さＡがゼロならば戻る
E0E7 47       MOV  B,A      Ｂに文字列変数の長さＡを移す（繰り返しの回数になる）
E0E8 23       INX  H        ＨＬをｓｄの２バイト目の番地にする
E0E9 CD E0F3  CALL E0F3     ＨＬに文字列変数の文字列の先頭番地を取り出す
                          文字列変数の１バイトを指定した番地にしまう
E0EC 1A       LDAX D        Ａに指定された番地の内容を取り出す
E0ED 77       MOV  M,A      指定された番地の内容Ａを文字列変数にしまう
                          繰り返しの指示をする
E0EE 13       INX  D        ＤＥを１つ先のメモリーの番地にする
E0EF 23       INX  H        ＨＬを文字列変数の次の１バイトの番地にする
E0F0 10 FA    DJNZ E0EC     残り処理バイト数Ｂから１を引きゼロでなければ繰り返す
E0F2 C9       RET         呼び出しプログラムに戻る
```

内部サブルーチン

機能　　　：ＨＬの指している番地から２バイトをＨＬに取り出す

入り口状態　：ＨＬ－値を取り出す先頭番地

戻り状態　　：ＨＬ－取り出した値

　　　　　　　ＨＬ以外のレジスターは変化しない

使用スタック：　１レベル

```
E0F3 D5       PUSH D        作業用レジスターＤＥの内容をスタックに保存する
E0F4 5E       MOV  E,M      ＨＬの指している番地の内容をＥに取り出す（下位バイト）
E0F5 23       INX  H        ＨＬを次のバイトの番地にする
E0F6 56       MOV  D,M      ＨＬの指している番地の内容をＤに取り出す（上位バイト）
E0F7 EB       XCHG          ＨＬに取り出した値をもってくる
E0F8 D1       POP  D        スタックから保存したＤＥの内容を取り戻す
E0F9 C9       RET           呼び出し機械語プログラムに戻る
```

☆（８８）機械語　例題６☆

１．機能

文字列に指定した内容をメモリーに書き込む。書き込む長さは文字列の長さとする。

２．呼び出し形式

ＰＯＫＥＬ＝＆ＨＥ１００
ＣＡＬＬ　ＰＯＫＥＬ（文字列変数，整数変数）
　文字列変数（入力）　：メモリーに書き込む内容（長さが書き込む長さ）
　整数変数（出力）　　：書き込むメモリーの開始番地

３．利用例

機械語例題５（ＰＥＥＫＬ）の機械語プログラムをメモリーに書き込む。機械語例題３（ＨＥＸＶＡＬ）も利用する。

```
100 HEXVAL=&HE060: POKEL=&HE100
110 H$="cdf3e0eb7eb7c84723cdf3e01a77132310fac9d55e2356ebd1c9"
120 V$=SPACE$(LEN(H$)/2): CALL HEXVAL(H$,V$)
130 A%=&HE0E0: CALL POKEL(V$,A%)
```

４．制約事項

・引き数の数や型が呼び出し形式と違う場合、正しく動かない。
・文字列変数が空文字列（ヌルストリング）の場合なにもしない。

５．機械語プログラム

入り口状態：ＨＬ－　文字列変数のストリングディスクリプター（ｓｄ）の先頭番地
　　　　　　ＤＥ－　整数変数の先頭番地

```
                          引数の情報を取り出す
E100 7E        MOV  A,M      Aに文字列変数の長さを取り出す
E101 B7        ORA  A        文字列変数の長さAがゼロかどうか調べる
E102 C8        RZ            文字列変数の長さAがゼロならば戻る
E103 47        MOV  B,A      Bに文字列変数の長さAを移す（繰り返しの回数になる）
E104 23        INX  H        HLをsdの2バイト目の番地にする
E105 CD E113   CALL E113     HLに文字列変数の文字列の先頭番地を取り出す
E108 EB        XCHG          HLに整数変数の先頭番地、DEに文字列の先頭番地を移す
E109 CD E113   CALL E113     HLに整数変数の値を取り出す
                          文字列変数の1バイトを指定した番地にしまう
E10C 1A        LDAX D        Aに文字列変数の1バイトを取り出す
E10D 77        MOV  M,A      文字列変数の1バイトAを指定した番地にしまう
                          繰り返しの指示をする
E10E 13        INX  D        DEを文字列変数の次の1バイトの番地にする
E10F 23        INX  H        HLを1つ先のメモリーの番地にする
E110 10 FA     DJNZ E10C     残り処理バイト数Bから1を引きゼロでなければ繰り返す
E112 C9        RET        呼び出しプログラムに戻る
```

内部サブルーチン
　機能　　　　：ＨＬの指している番地から２バイトをＨＬに取り出す
　入り口状態　：ＨＬ－値を取り出す先頭番地
　戻り状態　　：ＨＬ－取り出した値
　　　　　　　　ＨＬ以外のレジスターは変化しない
　使用スタック：　１レベル

```
E113 D5        PUSH D        作業用レジスターDEの内容をスタックに保存する
E114 5E        MOV  E,M      HLの指している番地の内容をEに取り出す（下位バイト）
E115 23        INX  H        HLを次のバイトの番地にする
E116 56        MOV  D,M      HLの指している番地の内容をDに取り出す（上位バイト）
E117 EB        XCHG          HLに取り出した値をもってくる
E118 D1        POP  D        スタックから保存したDEの内容を取り戻す
E119 C9        RET           呼び出し機械語プログラムに戻る
```

☆（８８）機械語　例題７☆

１．機能

文字列のなかにある特定の文字を別の文字に置き換える。引き数１は操作の対象となる文字列で処理結果も引き数１に戻る。引き数２と引き数３の文字列は先頭から１文字ずつ対応し、引き数２に置き換えの対象となる文字を指定する。引き数３は置き換える文字を指定する。

２．呼び出し形式

ＲＥＰＬＡＣＥ＝＆ＨＥ１２０
ＣＡＬＬ　ＲＥＰＬＡＣＥ（文字列変数１，文字列変数２，文字列変数３）

文字列変数１（入力／出力）	：処理の対象となる文字列
文字列変数２（入力）	：置き換えの対象となる文字（２文字以上指定した場合文字列変数３の先頭から１文字ずつ対応する）
文字列変数３（入力）	：置き換える文字（長さは文字列変数２と同じであること）

３．利用例

キーボードから入力した文字列のなかのアルファベットを大文字に変換して画面に表示する。また同じく小文字にして表示する。

```
100 REPLACE=&HE120
110 LC$="abcdefghijklmnopqrstuvwxyz"
120 UC$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
130 LINE INPUT "ﾓｼﾞﾚﾂ ﾊ? ";R$
140 CALL REPLACE(R$,LC$,UC$): PRINT R$
150 CALL REPLACE(R$,UC$,LC$): PRINT R$
run
ﾓｼﾞﾚﾂ ﾊ? Asahi Shimbun Denshi-keisan-shitsu
ASAHI SHIMBUN DENSHI-KEISAN-SHITSU
asahi shimbun denshi-keisan-shitsu
```

４．制約事項

・引き数の数や型が呼び出し形式と違う場合、正しく動かない。
・文字列変数２と文字列変数３の長さが違う場合は何もしない。
・引き数の中に空文字列（ヌルストリング）がある場合はなにもしない。
・文字列変数２の中に同じ文字がある場合は最初の文字が有効になる。

５．機械語プログラム

入り口状態：ＨＬ－　文字列変数１のストリングディスクリプター（ｓｄ１）の先頭番地
ＤＥ－　文字列変数２のストリングディスクリプター（ｓｄ２）の先頭番地
ＢＣ－　文字列変数３のストリングディスクリプター（ｓｄ３）の先頭番地

```
                                文字列変数１の長さをチェックし、引き数の文字列１を取り出す
E120 7E         MOV  A,M          Ａに文字列変数１の長さを取り出す
E121 B7         ORA  A            文字列変数１の長さＡがゼロかどうか調べる
E122 C8         RZ                長さがゼロならば戻る
E123 32 E173    STA  E173         文字列変数１の長さ（置き換え操作の繰り返し数）をしまう
E126 23         INX  H            ＨＬをｓｄ１の２バイト目の番地にする
E127 CD E16C    CALL E16C         ＨＬに文字列変数１の文字列の先頭番地を取り出す
E12A 22 E174    SHLD E174         文字列変数１の文字列の先頭番地をしまう
                                文字列変数２・３の長さをチェックする
E12D 0A         LDAX B            Ａに文字列変数３の長さを取り出す
E12E B7         ORA  A            文字列変数３の長さＡがゼロかどうか調べる
E12F C8         RZ                長さがゼロならば戻る
E130 EB         XCHG              ｓｄ１・ｓｄ２の番地をそれぞれＤＥ・ＨＬに移す
E131 BE         CMP  M            文字列変数３の長さＡと文字列変数２の長さを比べる
E132 C0         RNZ               長さが同じでなければ戻る
E133 32 E176    STA  E176         置き換えの対象となる文字数をしまう
                                文字列変数２・３の文字列の先頭番地を取り出す
E136 23         INX  H            ＨＬをｓｄ２の２バイト目の番地にする
E137 CD E16C    CALL E16C         ＨＬに文字列変数２の文字列の先頭番地を取り出す
```

```
E13A E5         PUSH H        スタックに文字列変数２の文字列の先頭番地をしまう
E13B 60         MOV  H,B      Ｈにｓｄ３の先頭番地の上位バイトを移す
E13C 69         MOV  L,C      Ｌにｓｄ３の先頭番地の下位バイトを移す
E13D 23         INX  H        ＨＬをｓｄ３の２バイト目の番地にする
E13E CD E16C    CALL E16C     ＨＬに文字列変数３の文字列の先頭番地を取り出す
E141 E5         PUSH H        スタックに文字列変数３の文字列の先頭番地をしまう
                            検索操作の繰り返しの初期設定をする
E142 3A E176    LDA  E176     Ａに置き換えの対象となる文字数を取り出す
E145 47         MOV  B,A      Ｂに置き換えの対象となる文字数Ａを移し繰り返し数とする
E146 2A E174    LHLD E174     文字列変数１の処理の対象となる文字の番地を取り出す
E149 7E         MOV  A,M      Ａに文字列変数１の処理の対象となる文字を取り出す
E14A D1         POP  D        ＤＥに文字列変数３の文字列の先頭番地を取り出す
E14B E1         POP  H        ＨＬに文字列変数２の文字列の先頭番地を取り出す
E14C E5         PUSH H        スタックに文字列変数２の文字列の先頭番地をしまいなおす
E14D D5         PUSH D        スタックに文字列変数３の文字列の先頭番地をしまいなおす
                            処理対象の文字Ａが文字列変数２のなかにあるかどうか調べる
E14E BE         CMP  M        処理対象の文字Ａと文字列変数２のなかの文字を比べる
E14F 28 06      JRZ  E157     同じ文字ならば置き換え操作へ行く
E151 23         INX  H        ＨＬを文字列変数２の次の文字の番地にする
E152 13         INX  D        ＤＥを文字列変数３の次の文字の番地にする
E153 10 F9      DJNZ E14E     置き換え対象の残り文字数Ｂから１つ引きゼロまで繰り返す
E155 18 01      JMPR E158     同じ文字が見つからないので置き換え処理をとばす
                            置き換え処理をする
E157 1A         LDAX D        Ａに文字列変数２に対応する文字列変数３の文字を取り出す
E158 2A E174    LHLD E174     文字列変数１の処理対象の文字の番地を取り出しなおす
E15B 77         MOV  M,A      文字列変数１の処理対象の文字を置き換える
                            置き換え操作の繰り返しを指示する
E15C 23         INX  H        文字列変数１の処理対象となる文字の番地を１つ進める
E15D 22 E174    SHLD E174     文字列変数１の処理対象となる文字の番地をしまう
E160 3A E173    LDA  E173     Ａに文字列変数１の未処理文字数を取り出す
E163 3D         DCR  A        文字列変数１の未処理文字数Ａから１つ引く
E164 32 E173    STA  E173     文字列変数１の未処理文字数Ａをしまう
E167 20 D9      JRNZ E142     文字列変数１の未処理文字数Ａがゼロでなければ繰り返す
                            スタックを整理して呼び出しプログラムに戻る
E169 D1         POP  D        スタックから文字列変数３の文字列の先頭番地を取り出す
E16A E1         POP  H        スタックから文字列変数２の文字列の先頭番地を取り出す
E16B C9         RET           呼び出しプログラムに戻る
```

内部サブルーチン

機能　　　：ＨＬの指している番地から２バイトをＨＬに取り出す
入り口状態：ＨＬ－値を取り出す先頭番地
戻り状態　：ＨＬ－取り出した値
　　　　　　ＨＬ以外のレジスターは変化しない
使用スタック：　１レベル

```
E16C D5         PUSH D        作業用レジスターＤＥの内容をスタックに保存する
E16D 5E         MOV  E,M      ＨＬの指している番地の内容をＥに取り出す（下位バイト）
E16E 23         INX  H        ＨＬを次のバイトの番地にする
E16F 56         MOV  D,M      ＨＬの指している番地の内容をＤに取り出す（上位バイト）
E170 EB         XCHG          ＨＬに取り出した値をもってくる
E171 D1         POP  D        スタックから保存したＤＥの内容を取り戻す
E172 C9         RET           呼び出し機械語プログラムに戻る
```

作業領域

```
E173            １バイト      文字列変数１の未処理文字数
E174            ２バイト      文字列変数１の処理対象文字の番地
E176            １バイト      置き換え対象文字数（文字列変数２・３の長さ）
```

エラーコード	N88－BAISC	意味
1	NEXT without FOR (NF)	FORがない。NEXTとFORが対応していない NEXTのあとの変数名に誤りはないか。FOR～NEXTのループの途中にGOTO文でとびこんでいないか
2	Syntax error (SN)	文法違反。ステートメントの記述に誤りがある LIST.やHELPキーでエラーの起きた行を調べる
3	RETURN without GOSUB (RG)	GOSUBでサブルーチンを呼ばないのにRETURNになった。GOTOでサブルーチンにとんだり、メインプログラムにENDがなくサブルーチンにつながっていないか
4	Out of DATA (OD)	READで読むデータが不足 READ文の変数とDATA文のデータの組み合わせを確認する。同じデータを再度読む時はRESTOREが必要
5	Illegal function call (FC)	コマンドや関数に与える数値パラメーターが指定できる範囲外の値。パラメーターの値を調べる
6	Overflow (OV)	数値や演算結果が範囲を越えた。変数が整数型でなければ、プログラムは停止せずメッセージを表示して継続
7	Out of memory (OM)	プログラムや配列が大きすぎてメモリー不足 BASICの使用できるメモリーの残り容量はFRE関数で分かる。使用するメモリーの大きさはCLEAR文で設定できる また、サブルーチンから自分自身を呼んでもメモリー不足になる場合がある
8	Undefined line number (UL)	プログラム行が定義されていない GOTO、GOSUBなど行番号を指定するステートメントで目的の番号がない。RENUMすると、とびさきのない行番号を表示する
9	Subscript out of range (BS)	配列変数の添字の値がDIMで定義した範囲外 変数名に誤りはないか、添字の値は範囲内か調べる
10	Duplicate Definition (DD)	同じ名前の配列を再定義した ERASEで配列を消去してから再定義する
11	Division by zero (／0)	0で割った。たとえば、未定義の変数で割った。変数が整数型でなければ、プログラムはメッセージを表示して継続

エラーコード	N88－BASIC	意　　味
12	Illegal direct (ID)	ダイレクトモードで使えないコマンドが入った
13	Type mismatch (TM)	式の左右で変数の型が合わない。コマンドや関数の引数の型がちがう。たとえば、数値変数に文字、文字変数に数値を代入
14	Out of string space (OS)	ストリング領域不足。ＣＬＥＡＲ文を調べよ
15	String too long (LS)	文字列を足した結果が２５５文字を越えた
16	String formula too complex (ST)	文字式が複雑すぎる
17	Can't continue (CN)	ＣＯＮＴできない。エラー停止やＳＴＯＰ後、プログラムを変更したり、消してしまったため。また、ＰＡＩＮＴの途中で止めてもＣＯＮＴできない
18	Undefined user function (UF)	ＤＥＦ文で定義していないユーザー関数を呼んだ ＤＥＦ文の名前がまちがっていないか
19	No RESUME (NR)	エラー処理に入ったが、ＲＥＳＵＭＥがない。エラー処理からの復帰はＲＥＳＵＭＥのみ
20	RESUME without error (RW)	エラー処理でないのにＲＥＳＵＭＥがあった ＯＮ　ＥＲＲＯＲ　ＧＯＴＯを設定してあるか
21	Unprintable error (UE)	エラーメッセージに定義されていないエラー
22	Missing operand (MO)	ステートメントの中に必要なパラメーターや変数がない 記述が途切れたりして不完全でないか調べる
23	Line buffer overflow (BO)	１行に入力できる文字の範囲を越えて入力した
26	FOR without NEXT (FN)	ＦＯＲに対応するＮＥＸＴがない
27	Tape read error (TP)	カセットテープの読み込みでエラーが発生した カセット装置のボリュームを調整してみる
29	WHILE without WEND (WH)	ＷＨＩＬＥに対応するＷＥＮＤがない
30	WEND without WHILE (WE)	ＷＨＩＬＥがない。ＷＥＮＤとＷＨＩＬＥが対応していない ＧＯＴＯ文でＷＨＩＬＥ～ＷＥＮＤのあいだにとんでいないか

エラーコード	N88−BASIC	意　　味
31	Duplicate label (DU)	同じラベル名が２つ以上定義されている。プログラムの実行直後にメッセージがでる
32	Undefined label (UN)	ラベル名が定義されていない ＧＯＴＯやＧＯＳＵＢのとびさきに誤りはないか
33	Feature not available (NA)	利用できない機能を指定した。ディスクＢＡＳＩＣの命令をディスクなしで使用していないか
50	FIELD overflow (FO)	ＦＩＥＬＤ文で合計２５７文字以上割り当てた
51	Internal error (IE)	ＢＡＳＩＣ内部でのエラー
52	Bad file number (BN)	システム立ち上げ時に指定したファイル番号を越えて使った ｈｏｗ　ｍａｎｙ　ｆｉｌｅｓの指定を思い出す
53	File not found (FF)	指定したファイルがフロッピーディスクになかった ＬＯＡＤ、ＫＩＬＬ、ＯＰＥＮ（ＩＮＰＵＴ、ＡＰＰＥＮＤ）などでファイルがない。ＦＩＬＥＳで確認する。フロッピーをまちがえていないか
54	File already open (AO)	すでにＯＰＥＮしてあるファイルを再度ＯＰＥＮしたり、ＫＩＬＬした。エラー終了した時はＣＬＯＳＥしないので注意 ダイレクトモードでＣＬＯＳＥと入れる
55	Input past end (EF)	シーケンシャルファイルのデータを読み終わった後にＩＮＰＵＴを実行、あるいはランダムファイルにないレコード番号を指定してＧＥＴを実行。シーケンシャルファイルではＥＯＦ、ランダムファイルではＬＯＦにより最大レコード番号を確認
56	Bad file name (NM)	不適当な文字をファイル名に使った。コロン（：）やコントロール文字は使えない。
57	Direct statement in file (DS)	プログラムのロードで、プログラムとはみなせないステートメントがあった。データファイルをＬＯＡＤしていないか
58	Sequential after PUT (SP)	ＰＵＴ文実行後、シーケンシャルファイルをアクセスしようとした。
59	Sequential I／O only (SQ)	シーケンシャル入出力以外できない
60	File not open (CF)	ＯＰＥＮしていないファイルに対してアクセスした。ファイル番号に誤りはないか

エラーコード	N88-BASIC 意味
61	File write protected ライトプロテクトがかかっているファイルに対して書き込んだり、削除しようとした。プロテクトの確認はATTR$、解除はSET
62	Disk offline フロッピーディスクの入っていないディスク装置にアクセスした。ドライブ番号の誤りはないか。ディスクのふたが空いていないか
64	Disk I/O error フロッピーディスクの不良で、読み書き時にエラーが発生 大切なファイルの入ったフロッピーディスクはバックアップを作っておく
65	File already exists NAME文により新しくつけなおすファイル名がすでに登録されている
68	Disk full ディスク上のすべての場所を使いきった DSKFで空き容量を確認しておく。PUTで大きいレコード番号を指定すると、1からその番号までのレコードが入るファイルができるので注意
69	Bad allocation table FATがこわれた。FATにDSKO$で書き込みをしなかったか
70	Bad drive number ディスク装置のつながっていないドライブ番号にアクセスした
71	Bad track/sector DSKI$、DSKO$でサーフェス番号、トラック番号、セクター番号の指定が範囲外
72	Deleted record 消去したレコードをアクセスした
73	Rename across disks ファイル名の変更で旧ファイル名と新ファイル名のドライブ番号がちがう
74	Illegal operation キーボード操作が正しくない。

2. キャラクターコード表

16進	0	1	2	3	4	5	6	7
0	0	DE 16	32	0 48	@ 64	P 80	96	p 112
1	SH 1	D1 17	! 33	1 49	A 65	Q 81	a 97	q 113
2	SX 2	D2 18	” 34	2 50	B 66	R 82	b 98	r 114
3	EX 3	D3 19	# 35	3 51	C 67	S 83	c 99	s 115
4	ET 4	D4 20	$ 36	4 52	D 68	T 84	d 100	t 116
5	EQ 5	NK 21	% 37	5 53	E 69	U 85	e 101	u 117
6	AK 6	SN 22	& 38	6 54	F 70	V 86	f 102	v 118
7	BL 7	EB 23	’ 39	7 55	G 71	W 87	g 103	w 119
8	BS 8	CN 24	(40	8 56	H 72	X 88	h 104	x 120
9	HT 9	EM 25	) 41	9 57	I 73	Y 89	i 105	y 121
A	LF 10	SB 26	* 42	: 58	J 74	Z 90	j 106	z 122
B	HM 11	EC 27	+ 43	; 59	K 75	[91	k 107	{ 123
C	CL 12	→ 28	, 44	< 60	L 76	¥ 92	l 108	: 124
D	CR 13	← 29	— 45	= 61	M 77	] 93	m 109	} 125
E	SO 14	↑ 30	. 46	> 62	N 78	^ 94	n 110	~ 126
F	SI 15	↓ 31	/ 47	? 63	O 79	_ 95	o 111	127

１６進数は横が上位４ビット縦が下位４ビット
マスの中の数字は１６進数に対応する１０進数（たとえば４ＢはＫだが１０進数では７５になる）

8	9	A	B	C	D	E	F	16進
▁ 128	┴ 144	160	ｰ 176	ﾀ 192	ﾐ 208	═ 224	╳ 240	0
▂ 129	┬ 145	｡ 161	ｱ 177	ﾁ 193	ﾑ 209	╞ 225	円 241	1
▃ 130	┤ 146	｢ 162	ｲ 178	ﾂ 194	ﾒ 210	╪ 226	年 242	2
▄ 131	├ 147	｣ 163	ｳ 179	ﾃ 195	ﾓ 211	╡ 227	月 243	3
▅ 132	▔ 148	､ 164	ｴ 180	ﾄ 196	ﾔ 212	◢ 228	日 244	4
▆ 133	─ 149	･ 165	ｵ 181	ﾅ 197	ﾕ 213	◣ 229	時 245	5
▇ 134	│ 150	ｦ 166	ｶ 182	ﾆ 198	ﾖ 214	◥ 230	分 246	6
█ 135	▕ 151	ｧ 167	ｷ 183	ﾇ 199	ﾗ 215	◤ 231	秒 247	7
▏ 136	┌ 152	ｨ 168	ｸ 184	ﾈ 200	ﾘ 216	♠ 232	248	8
▎ 137	┐ 153	ｩ 169	ｹ 185	ﾉ 201	ﾙ 217	♥ 233	249	9
▍ 138	└ 154	ｪ 170	ｺ 186	ﾊ 202	ﾚ 218	♦ 234	250	A
▌ 139	┘ 155	ｫ 171	ｻ 187	ﾋ 203	ﾛ 219	♣ 235	251	B
▋ 140	╭ 156	ｬ 172	ｼ 188	ﾌ 204	ﾜ 220	● 236	252	C
▊ 141	╮ 157	ｭ 173	ｽ 189	ﾍ 205	ﾝ 221	○ 237	253	D
▉ 142	╰ 158	ｮ 174	ｾ 190	ﾎ 206	ﾞ 222	╱ 238	254	E
┼ 143	╯ 159	ｯ 175	ｿ 191	ﾏ 207	ﾟ 223	╲ 239	255	F

3. 日本語JISコード表

テキスト画面に表示できる半角文字一覧表　　　　(156字)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
002		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
003	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
004	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
005	P	Q	R	S	T	U	V	W	X	Y	Z	[	¥	]	^	—
006		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
007	p	q	r	s	t	u	v	w	x	y	z	{	¦	}	~	
008		。	「	」		・	ヲ	ァ	ィ	ゥ	ェ	ォ	ャ	ュ	ョ	ッ
00B	ー	ア	イ	ウ	エ	オ	カ	キ	ク	ケ	コ	サ	シ	ス	セ	ソ
00C	タ	チ	ツ	テ	ト	ナ	ニ	ヌ	ネ	ノ	ハ	ヒ	フ	ヘ	ホ	マ
00D	ミ	ム	メ	モ	ヤ	ユ	ヨ	ラ	リ	ル	レ	ロ	ワ	ン	゛	゜

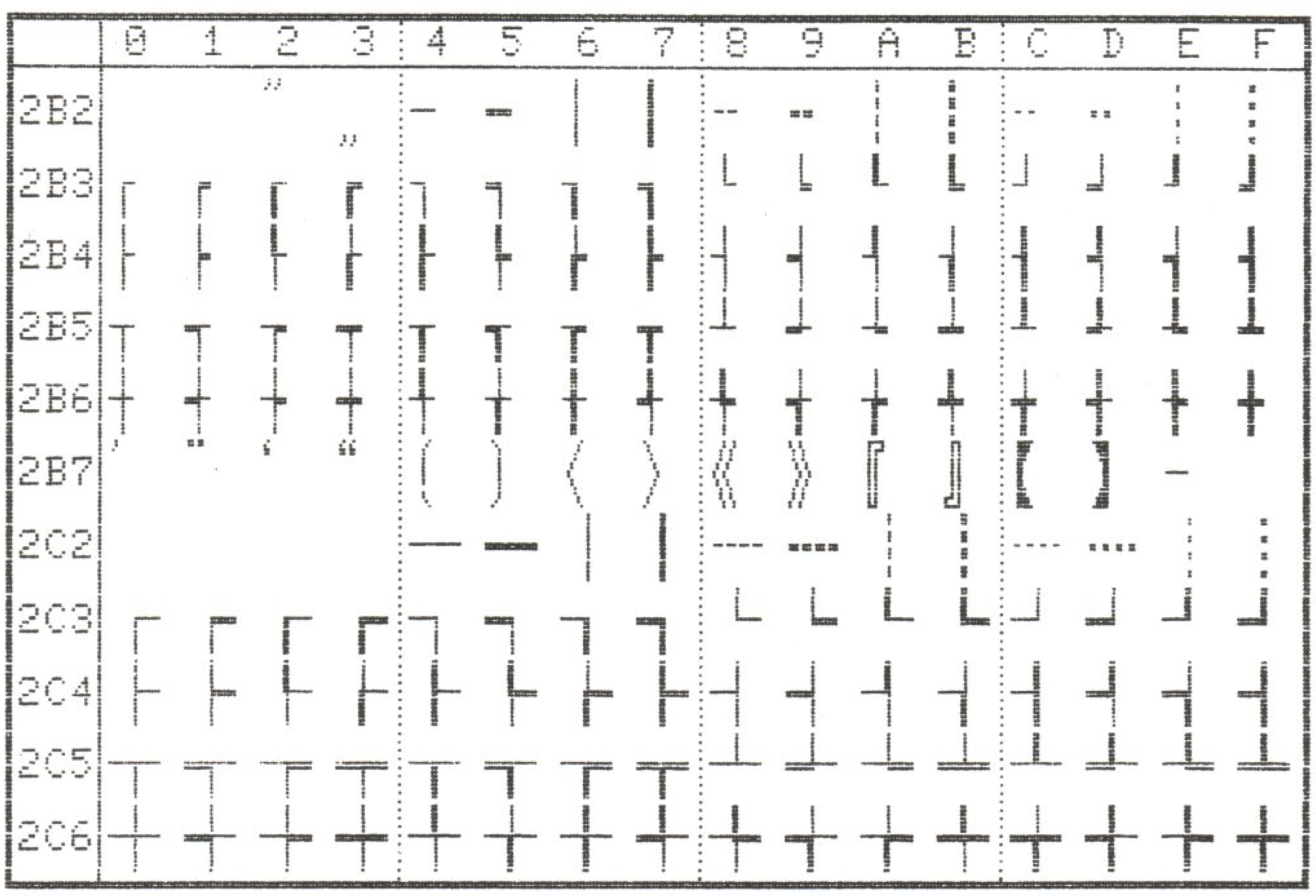
グラフィック画面に表示できる半角文字一覧表　　　　(219字)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
002		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
003	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
004	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
005	P	Q	R	S	T	U	V	W	X	Y	Z	[	¥	]	^	—
006		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
007	p	q	r	s	t	u	v	w	x	y	z	{	¦	}	~	
008		。	「	」	、	・	を	ぁ	ぃ	ぅ	ぇ	ぉ	ゃ	ゅ	ょ	っ
009	ー	あ	い	う	え	お	か	き	く	け	こ	さ	し	す	せ	そ
00A		。	「	」		・	ヲ	ァ	ィ	ゥ	ェ	ォ	ャ	ュ	ョ	ッ
00B	ー	ア	イ	ウ	エ	オ	カ	キ	ク	ケ	コ	サ	シ	ス	セ	ソ
00C	タ	チ	ツ	テ	ト	ナ	ニ	ヌ	ネ	ノ	ハ	ヒ	フ	ヘ	ホ	マ
00D	ミ	ム	メ	モ	ヤ	ユ	ヨ	ラ	リ	ル	レ	ロ	ワ	ン	゛	゜
00E	た	ち	つ	て	と	な	に	ぬ	ね	の	は	ひ	ふ	へ	ほ	ま
00F	み	む	め	も	や	ゆ	よ	ら	り	る	れ	ろ	わ	ん	゛	゜

グラフィック画面に表示できる¼角文字一覧表　　　　　　(213字)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
010		S_H	S_X	E_X	E_T	E_Q	A_K	B_L	B_S	H_T	L_F	H_M	C_L	C_R	S_O	S_I
011	D_E	D_1	D_2	D_3	D_4	N_K	S_N	E_B	C_N	E_M	S_B	E_C	→	←	↑	↓
012		!	"	#	$	%	&	'	(	)	*	+	,	−	.	/
013	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
014	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
015	P	Q	R	S	T	U	V	W	X	Y	Z	[	¥	]	^	—
016		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
017	p	q	r	s	t	u	v	w	x	y	z	{	¦	}	~	
018	▁	▂	▃	▄	▅	▆	▇	█	▏	▎	▍	▌	▋	▊	▉	┼
019	┴	┬	┤	├	▔	─	│	▕	┌	┐	└	┘	╭	╮	╰	╯
01A		。	「	」	、	・	ヲ	ァ	ィ	ゥ	ェ	ォ	ャ	ュ	ョ	ッ
01B	ー	ア	イ	ウ	エ	オ	カ	キ	ク	ケ	コ	サ	シ	ス	セ	ソ
01C	タ	チ	ツ	テ	ト	ナ	ニ	ヌ	ネ	ノ	ハ	ヒ	フ	ヘ	ホ	マ
01D	ミ	ム	メ	モ	ヤ	ユ	ヨ	ラ	リ	ル	レ	ロ	ワ	ン	゛	゜
01E	═	╞	╪	╡	◢	◣	◥	◤	♠	♥	♦	♣	●	○	╱	╲
01F	×	円	年	月	日	時	分	秒								

ケイ線文字



漢字コード表

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
212			、	。	，	．	・	：	；	？	！	゛	゜	´	｀	¨
214	＼	～	∥	｜	…	‥	‘	’	“	”	（	）	〔	〕	［	］
216	÷	＝	≠	＜	＞	≦	≧	∞	∴	♂	♀	°	′	″	℃	￥
222		◆	□	■	△	▲	▽	▼	※	〒	→	←	↑	↓	＝	
232																
234		Ａ	Ｂ	Ｃ	Ｄ	Ｅ	Ｆ	Ｇ	Ｈ	Ｉ	Ｊ	Ｋ	Ｌ	Ｍ	Ｎ	Ｏ
236		ａ	ｂ	ｃ	ｄ	ｅ	ｆ	ｇ	ｈ	ｉ	ｊ	ｋ	ｌ	ｍ	ｎ	ｏ
242		ぁ	あ	ぃ	い	ぅ	う	ぇ	え	ぉ	お	か	が	き	ぎ	く
244	だ	ち	ぢ	っ	つ	づ	て	で	と	ど	な	に	ぬ	ね	の	は
246	む	め	も	ゃ	や	ゅ	ゆ	ょ	よ	ら	り	る	れ	ろ	ゎ	わ
252		ァ	ア	ィ	イ	ゥ	ウ	ェ	エ	ォ	オ	カ	ガ	キ	ギ	ク
254	ダ	チ	ヂ	ッ	ツ	ヅ	テ	デ	ト	ド	ナ	ニ	ヌ	ネ	ノ	ハ
256	ム	メ	モ	ャ	ヤ	ュ	ユ	ョ	ヨ	ラ	リ	ル	レ	ロ	ヮ	ワ
262		Α	Β	Γ	Δ	Ε	Ζ	Η	Θ	Ι	Κ	Λ	Μ	Ν	Ξ	Ο
264		α	β	γ	δ	ε	ζ	η	θ	ι	κ	λ	μ	ν	ξ	ο
272		А	Б	В	Г	Д	Е	Ё	Ж	З	И	Й	К	Л	М	Н
274	Ю	Я														
276	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э
302		亜	唖	娃	阿	哀	愛	挨	姶	逢	葵	茜	穐	悪	握	渥
304	粟	袷	安	庵	按	暗	案	闇	鞍	杏	以	伊	位	依	偉	囲
306	萎	衣	謂	違	遺	医	井	亥	域	育	郁	磯	一	壱	溢	逸
312		院	陰	隠	韻	吋	右	宇	烏	羽	迂	雨	卯	鵜	窺	丑
314	雲	荏	餌	叡	営	嬰	影	映	曳	栄	永	泳	洩	瑛	盈	穎
316	園	堰	奄	宴	延	怨	掩	援	沿	演	炎	焔	煙	燕	猿	縁
322		押	旺	横	欧	殴	王	翁	襖	鶯	鷗	黄	岡	沖	荻	億
324	伽	価	佳	加	可	嘉	夏	嫁	家	寡	科	暇	果	架	歌	河
326	迦	過	霞	蚊	俄	峨	我	牙	画	臥	芽	蛾	賀	雅	餓	駕
332		魁	晦	械	海	灰	界	皆	絵	芥	蟹	開	階	貝	凱	劾
334	垣	柿	蠣	鈎	劃	嚇	各	廓	拡	攪	格	核	殻	獲	確	穫
336	橿	梶	鰍	潟	割	喝	恰	括	活	渇	滑	葛	褐	轄	且	鰹
342		粥	刈	苅	瓦	乾	侃	冠	寒	刊	勘	勧	巻	喚	堪	姦
344	汗	漢	澗	灌	環	甘	監	看	竿	管	簡	緩	缶	翰	肝	艦
346	巌	玩	癌	眼	岩	翫	贋	雁	頑	顔	願	企	伎	危	喜	器
352		機	帰	毅	気	汽	畿	祈	季	稀	紀	徽	規	記	貴	起
354	祇	義	蟻	誼	議	掬	菊	鞠	吉	吃	喫	桔	橘	詰	砧	杵
356	朽	求	汲	泣	灸	球	究	窮	笈	級	糾	給	旧	牛	去	居
362		供	侠	僑	兇	競	共	凶	協	匡	卿	叫	喬	境	峡	強
364	鏡	響	饗	驚	仰	凝	堯	暁	業	局	曲	極	玉	桐	粁	僅
366	謹	近	金	吟	銀	九	倶	句	区	狗	玖	矩	苦	躯	駆	駈
372		掘	窟	沓	靴	轡	窪	熊	隈	粂	栗	繰	桑	鍬	勲	君
374	契	形	径	恵	慶	慧	憩	掲	携	敬	景	桂	渓	畦	稽	系
376	劇	戟	撃	激	隙	桁	傑	欠	決	潔	穴	結	血	訣	月	件
382		検	権	牽	犬	献	研	硯	絹	県	肩	見	謙	賢	軒	遣
384	言	諺	限	乎	個	古	呼	固	姑	孤	己	庫	弧	戸	故	枯
386	伍	午	呉	吾	娯	後	御	悟	梧	檎	瑚	碁	語	誤	護	醐
392		后	喉	坑	垢	好	孔	孝	宏	工	巧	巷	幸	広	庚	康
394	浩	港	溝	甲	皇	硬	稿	糠	紅	紘	絞	綱	耕	考	肯	肱
396	項	香	高	鴻	剛	劫	号	合	壕	拷	濠	豪	轟	麹	克	刻
3A2		此	頃	今	困	坤	墾	婚	恨	懇	昏	昆	根	梱	混	痕
3A4	裟	坐	座	挫	債	催	再	最	哉	塞	妻	宰	彩	才	採	栽
3A6	材	罪	財	冴	坂	阪	堺	榊	肴	咲	崎	埼	碕	鷺	作	削
3B2		察	拶	撮	擦	札	殺	薩	雑	皐	鯖	捌	錆	鮫	皿	晒
3B4	酸	餐	斬	暫	残	仕	仔	伺	使	刺	司	史	嗣	四	士	始
3B6	死	氏	獅	祉	私	糸	紙	紫	肢	脂	至	視	詞	詩	試	誌
3C2		次	滋	治	爾	璽	痔	磁	示	而	耳	自	蒔	辞	汐	鹿
3C4	疾	質	実	蔀	篠	偲	柴	芝	屡	蘂	縞	舎	写	射	捨	赦
3C6	酌	釈	錫	若	寂	弱	惹	主	取	守	手	朱	殊	狩	珠	種
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
213	＾	￣	＿	ヽ	ヾ	ゝ	ゞ	〃	仝	々	〆	〇	ー	―	‐	／
215	｛	｝	〈	〉	《	》	「	」	『	』	【	】	＋	－	±	×
217	＄	￠	￡	％	＃	＆	＊	＠	§	☆	★	○	●	◎	◇	
223																
233	０	１	２	３	４	５	６	７	８	９						
235	Ｐ	Ｑ	Ｒ	Ｓ	Ｔ	Ｕ	Ｖ	Ｗ	Ｘ	Ｙ	Ｚ					
237	ｐ	ｑ	ｒ	ｓ	ｔ	ｕ	ｖ	ｗ	ｘ	ｙ	ｚ					
243	ぐ	け	げ	こ	ご	さ	ざ	し	じ	す	ず	せ	ぜ	そ	ぞ	た
245	ば	ぱ	ひ	び	ぴ	ふ	ぶ	ぷ	へ	べ	ぺ	ほ	ぼ	ぽ	ま	み
247	ゐ	ゑ	を	ん												
253	グ	ケ	ゲ	コ	ゴ	サ	ザ	シ	ジ	ス	ズ	セ	ゼ	ソ	ゾ	タ
255	バ	パ	ヒ	ビ	ピ	フ	ブ	プ	ヘ	ベ	ペ	ホ	ボ	ポ	マ	ミ
257	ヰ	ヱ	ヲ	ン	ヴ	ヵ	ヶ									
263	Π	Ρ	Σ	Τ	Υ	Φ	Χ	Ψ	Ω							
265	π	ρ	σ	τ	υ	φ	χ	ψ	ω							
273	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э
275		а	б	в	г	д	е	ё	ж	з	и	й	к	л	м	н
277	ю	я														
303	旭	葦	芦	鯵	梓	圧	斡	扱	宛	姐	虻	飴	絢	綾	鮎	或
305	夷	委	威	尉	惟	意	慰	易	椅	為	畏	異	移	維	緯	胃
307	稲	茨	芋	鰯	允	印	咽	員	因	姻	引	飲	淫	胤	蔭	
313	碓	臼	渦	嘘	唄	欝	蔚	鰻	姥	厩	浦	瓜	閏	噂	云	運
315	頴	英	衛	詠	鋭	液	疫	益	駅	悦	謁	越	閲	榎	厭	円
317	艶	苑	薗	遠	鉛	鴛	塩	於	汚	甥	凹	央	奥	往	応	
323	屋	憶	臆	桶	牡	乙	俺	卸	恩	温	穏	音	下	化	仮	何
325	火	珂	禍	禾	稼	箇	花	苛	茄	荷	華	菓	蝦	課	嘩	貨
327	介	会	解	回	塊	壊	廻	快	怪	悔	恢	懐	戒	拐	改	
333	外	咳	害	崖	慨	概	涯	碍	蓋	街	該	鎧	骸	浬	馨	蛙
335	覚	角	赫	較	郭	閣	隔	革	学	岳	楽	額	顎	掛	笠	樫
337	叶	椛	樺	鞄	株	兜	竃	蒲	釜	鎌	噛	鴨	栢	茅	萱	
343	完	官	寛	干	幹	患	感	慣	憾	換	敢	柑	桓	棺	款	歓
345	莞	観	諌	貫	還	鑑	間	閑	関	陥	韓	館	舘	丸	含	岸
347	基	奇	嬉	寄	岐	希	幾	忌	揮	机	旗	既	期	棋	棄	
353	軌	輝	飢	騎	鬼	亀	偽	儀	妓	宜	戯	技	擬	欺	犠	疑
355	黍	却	客	脚	虐	逆	丘	久	仇	休	及	吸	宮	弓	急	救
357	巨	拒	拠	挙	渠	虚	許	距	鋸	漁	禦	魚	亨	享	京	
363	彊	怯	恐	恭	挟	教	橋	況	狂	狭	矯	胸	脅	興	蕎	郷
365	勤	均	巾	錦	斤	欣	欽	琴	禁	禽	筋	緊	芹	菌	衿	襟
367	駒	具	愚	虞	喰	空	偶	寓	遇	隅	串	櫛	釧	屑	屈	
373	薫	訓	群	軍	郡	卦	袈	祁	係	傾	刑	兄	啓	圭	珪	型
375	経	継	繋	罫	茎	荊	蛍	計	詣	警	軽	頚	鶏	芸	迎	鯨
377	倹	倦	健	兼	券	剣	喧	圏	堅	嫌	建	憲	懸	拳	捲	
383	鍵	険	顕	験	鹸	元	原	厳	幻	弦	減	源	玄	現	絃	舷
385	湖	狐	糊	袴	股	胡	菰	虎	誇	跨	鈷	雇	顧	鼓	五	互
387	乞	鯉	交	佼	侯	候	倖	光	公	功	効	勾	厚	口	向	
393	弘	恒	慌	抗	拘	控	攻	昂	晃	更	杭	校	梗	構	江	洪
395	腔	膏	航	荒	行	衡	講	貢	購	郊	酵	鉱	砿	鋼	閤	降
397	告	国	穀	酷	鵠	黒	獄	漉	腰	甑	忽	惚	骨	狛	込	
3A3	紺	艮	魂	些	佐	叉	唆	嵯	左	差	査	沙	瑳	砂	詐	鎖
3A5	歳	済	災	采	犀	砕	砦	祭	斎	細	菜	裁	載	際	剤	在
3A7	咋	搾	昨	朔	柵	窄	策	索	錯	桜	鮭	笹	匙	冊	刷	
3B3	三	傘	参	山	惨	撒	散	桟	燦	珊	産	算	纂	蚕	讃	賛
3B5	姉	姿	子	屍	市	師	志	思	指	支	孜	斯	施	旨	枝	止
3B7	諮	資	賜	雌	飼	歯	事	似	侍	児	字	寺	慈	持	時	
3C3	式	識	鴫	竺	軸	宍	雫	七	叱	執	失	嫉	室	悉	湿	漆
3C5	斜	煮	社	紗	者	謝	車	遮	蛇	邪	借	勺	尺	杓	灼	爵
3C7	腫	趣	酒	首	儒	受	呪	寿	授	樹	綬	需	囚	収	周	
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
3D2		宗	就	州	修	愁	拾	洲	秀	秋	終	繍	習	臭	舟	蒐	3D3	衆	襲	讐	蹴	輯	週	酋	酬	集	醜	什	住	充	十	従	戎
3D4	柔	汁	渋	獣	縦	重	銃	叔	夙	宿	淑	祝	縮	粛	塾	熟	3D5	出	術	述	俊	峻	春	瞬	竣	舜	駿	准	循	旬	楯	殉	淳
3D6	準	潤	盾	純	巡	遵	醇	順	処	初	所	暑	曙	渚	庶	緒	3D7	署	書	薯	藷	諸	助	叙	女	序	徐	恕	鋤	除	傷	償	
3E2		勝	匠	升	召	哨	商	唱	嘗	奨	妾	娼	宵	将	小	少	3E3	尚	庄	床	廠	彰	承	抄	招	掌	捷	昇	昌	昭	晶	松	梢
3E4	樟	樵	沼	消	渉	湘	焼	焦	照	症	省	硝	礁	祥	称	章	3E5	笑	粧	紹	肖	菖	蒋	蕉	衝	裳	訟	証	詔	詳	象	賞	醤
3E6	鉦	鍾	鐘	障	鞘	上	丈	丞	乗	冗	剰	城	場	壌	嬢	常	3E7	情	擾	条	杖	浄	状	畳	穣	蒸	譲	醸	錠	嘱	埴	飾	
3F2		拭	植	殖	燭	織	職	色	触	食	蝕	辱	尻	伸	信	侵	3F3	唇	娠	寝	審	心	慎	振	新	晋	森	榛	浸	深	申	疹	真
3F4	神	秦	紳	臣	芯	薪	親	診	身	辛	進	針	震	人	仁	刃	3F5	塵	壬	尋	甚	尽	腎	訊	迅	陣	靭	笥	諏	須	酢	図	厨
3F6	逗	吹	垂	帥	推	水	炊	睡	粋	翠	衰	遂	酔	錐	錘	随	3F7	瑞	髄	崇	嵩	数	枢	趨	雛	据	杉	椙	菅	頗	雀	裾	
402		澄	摺	寸	世	瀬	畝	是	凄	制	勢	姓	征	性	成	政	403	整	星	晴	棲	栖	正	清	牲	生	盛	精	聖	声	製	西	誠
404	誓	請	逝	醒	青	静	斉	税	脆	隻	席	惜	戚	斥	昔	析	405	石	積	籍	績	脊	責	赤	跡	蹟	碩	切	拙	接	摂	折	設
406	窃	節	説	雪	絶	舌	蝉	仙	先	千	占	宣	専	尖	川	戦	407	扇	撰	栓	栴	泉	浅	洗	染	潜	煎	煽	旋	穿	箭	線	
412		繊	羨	腺	舛	船	薦	詮	賎	践	選	遷	銭	銑	閃	鮮	413	前	善	漸	然	全	禅	繕	膳	糎	噌	塑	岨	措	曾	曽	楚
414	狙	疏	疎	礎	祖	租	粗	素	組	蘇	訴	阻	遡	鼠	僧	創	415	双	叢	倉	喪	壮	奏	爽	宋	層	匝	惣	想	捜	掃	挿	掻
416	操	早	曹	巣	槍	槽	漕	燥	争	痩	相	窓	糟	総	綜	聡	417	草	荘	葬	蒼	藻	装	走	送	遭	鎗	霜	騒	像	増	憎	
422		臓	蔵	贈	造	促	側	則	即	息	捉	束	測	足	速	俗	423	属	賊	族	続	卒	袖	其	揃	存	孫	尊	損	村	遜	他	多
424	太	汰	詑	唾	堕	妥	惰	打	柁	舵	楕	陀	駄	騨	体	堆	425	対	耐	岱	帯	待	怠	態	戴	替	泰	滞	胎	腿	苔	袋	貸
426	退	逮	隊	黛	鯛	代	台	大	第	醍	題	鷹	滝	瀧	卓	啄	427	宅	托	択	拓	沢	濯	琢	託	鐸	濁	諾	茸	凧	蛸	只	
432		叩	但	達	辰	奪	脱	巽	竪	辿	棚	谷	狸	鱈	樽	誰	433	丹	単	嘆	坦	担	探	旦	歎	淡	湛	炭	短	端	箪	綻	耽
434	胆	蛋	誕	鍛	団	壇	弾	断	暖	檀	段	男	談	値	知	地	435	弛	恥	智	池	痴	稚	置	致	蜘	遅	馳	築	畜	竹	筑	蓄
436	逐	秩	窒	茶	嫡	着	中	仲	宙	忠	抽	昼	柱	注	虫	衷	437	註	酎	鋳	駐	樗	瀦	猪	苧	著	貯	丁	兆	凋	喋	寵	
442		帖	帳	庁	弔	張	彫	徴	懲	挑	暢	朝	潮	牒	町	眺	443	聴	脹	腸	蝶	調	諜	超	跳	銚	長	頂	鳥	勅	捗	直	朕
444	沈	珍	賃	鎮	陳	津	墜	椎	槌	追	鎚	痛	通	塚	栂	掴	445	槻	佃	漬	柘	辻	蔦	綴	鍔	椿	潰	坪	壷	嬬	紬	爪	吊
446	釣	鶴	亭	低	停	偵	剃	貞	呈	堤	定	帝	底	庭	廷	弟	447	悌	抵	挺	提	梯	汀	碇	禎	程	締	艇	訂	諦	蹄	逓	
452		邸	鄭	釘	鼎	泥	摘	擢	敵	滴	的	笛	適	鏑	溺	哲	453	徹	撤	轍	迭	鉄	典	填	天	展	店	添	纏	甜	貼	転	顛
454	点	伝	殿	澱	田	電	兎	吐	堵	塗	妬	屠	徒	斗	杜	渡	455	登	菟	賭	途	都	鍍	砥	砺	努	度	土	奴	怒	倒	党	冬
456	凍	刀	唐	塔	塘	套	宕	島	嶋	悼	投	搭	東	桃	梼	棟	457	盗	淘	湯	涛	灯	燈	当	痘	祷	等	答	筒	糖	統	到	
462		董	蕩	藤	討	謄	豆	踏	逃	透	鐙	陶	頭	騰	闘	働	463	動	同	堂	導	憧	撞	洞	瞳	童	胴	萄	道	銅	峠	鴇	匿
464	得	徳	涜	特	督	禿	篤	毒	独	読	栃	橡	凸	突	椴	届	465	鳶	苫	寅	酉	瀞	噸	屯	惇	敦	沌	豚	遁	頓	呑	曇	鈍
466	奈	那	内	乍	凪	薙	謎	灘	捺	鍋	楢	馴	縄	畷	南	楠	467	軟	難	汝	二	尼	弐	迩	匂	賑	肉	虹	廿	日	乳	入	
472		如	尿	韮	任	妊	忍	認	濡	禰	祢	寧	葱	猫	熱	年	473	念	捻	撚	燃	粘	乃	廼	之	埜	嚢	悩	濃	納	能	脳	膿
474	農	覗	蚤	巴	把	播	覇	杷	波	派	琶	破	婆	罵	芭	馬	475	俳	廃	拝	排	敗	杯	盃	牌	背	肺	輩	配	倍	培	媒	梅
476	楳	煤	狽	買	売	賠	陪	這	蝿	秤	矧	萩	伯	剥	博	拍	477	柏	泊	白	箔	粕	舶	薄	迫	曝	漠	爆	縛	莫	駁	麦	
482		函	箱	硲	箸	肇	筈	櫨	幡	肌	畑	畠	八	鉢	溌	発	483	醗	髪	伐	罰	抜	筏	閥	鳩	噺	塙	蛤	隼	伴	判	半	反
484	叛	帆	搬	斑	板	氾	汎	版	犯	班	畔	繁	般	藩	販	範	485	釆	煩	頒	飯	挽	晩	番	盤	磐	蕃	蛮	匪	卑	否	妃	庇
486	彼	悲	扉	批	披	斐	比	泌	疲	皮	碑	秘	緋	罷	肥	被	487	誹	費	避	非	飛	樋	簸	備	尾	微	枇	毘	琵	眉	美	
492		鼻	柊	稗	匹	疋	髭	彦	膝	菱	肘	弼	必	畢	筆	逼	493	桧	姫	媛	紐	百	謬	俵	彪	標	氷	漂	瓢	票	表	評	豹
494	廟	描	病	秒	苗	錨	鋲	蒜	蛭	鰭	品	彬	斌	浜	瀕	貧	495	賓	頻	敏	瓶	不	付	埠	夫	婦	富	冨	布	府	怖	扶	敷
496	斧	普	浮	父	符	腐	膚	芙	譜	負	賦	赴	阜	附	侮	撫	497	武	舞	葡	蕪	部	封	楓	風	葺	蕗	伏	副	復	幅	服	
4A2		福	腹	複	覆	淵	弗	払	沸	仏	物	鮒	分	吻	噴	墳	4A3	憤	扮	焚	奮	粉	糞	紛	雰	文	聞	丙	併	兵	塀	幣	平
4A4	弊	柄	並	蔽	閉	陛	米	頁	僻	壁	癖	碧	別	瞥	蔑	箆	4A5	偏	変	片	篇	編	辺	返	遍	便	勉	娩	弁	鞭	保	舗	鋪
4A6	圃	捕	歩	甫	補	輔	穂	募	墓	慕	戊	暮	母	簿	菩	倣	4A7	俸	包	呆	報	奉	宝	峰	峯	崩	庖	抱	捧	放	方	朋	
4B2		法	泡	烹	砲	縫	胞	芳	萌	蓬	蜂	褒	訪	豊	邦	鋒	4B3	飽	鳳	鵬	乏	亡	傍	剖	坊	妨	帽	忘	忙	房	暴	望	某
4B4	棒	冒	紡	肪	膨	謀	貌	貿	鉾	防	吠	頬	北	僕	卜	墨	4B5	撲	朴	牧	睦	穆	釦	勃	没	殆	堀	幌	奔	本	翻	凡	盆
4B6	摩	磨	魔	麻	埋	妹	昧	枚	毎	哩	槙	幕	膜	枕	鮪	柾	4B7	鱒	桝	亦	俣	又	抹	末	沫	迄	侭	繭	麿	万	慢	満	
4C2		漫	蔓	味	未	魅	巳	箕	岬	密	蜜	湊	蓑	稔	脈	妙	4C3	粍	民	眠	務	夢	無	牟	矛	霧	鵡	椋	婿	娘	冥	名	命
4C4	明	盟	迷	銘	鳴	姪	牝	滅	免	棉	綿	緬	面	麺	摸	模	4C5	茂	妄	孟	毛	猛	盲	網	耗	蒙	儲	木	黙	目	杢	勿	餅
4C6	尤	戻	籾	貰	問	悶	紋	門	匁	也	冶	夜	爺	耶	野	弥	4C7	矢	厄	役	約	薬	訳	躍	靖	柳	薮	鑓	愉	愈	油	癒	
4D2		諭	輸	唯	佑	優	勇	友	宥	幽	悠	憂	揖	有	柚	湧	4D3	涌	猶	猷	由	祐	裕	誘	遊	邑	郵	雄	融	夕	予	余	与
4D4	誉	輿	預	傭	幼	妖	容	庸	揚	揺	擁	曜	楊	様	洋	溶	4D5	熔	用	窯	羊	耀	葉	蓉	要	謡	踊	遥	陽	養	慾	抑	欲
4D6	沃	浴	翌	翼	淀	羅	螺	裸	来	莱	頼	雷	洛	絡	落	酪	4D7	乱	卵	嵐	欄	濫	藍	蘭	覧	利	吏	履	李	梨	理	璃	
4E2		痢	裏	裡	里	離	陸	律	率	立	葎	掠	略	劉	流	溜	4E3	琉	留	硫	粒	隆	竜	龍	侶	慮	旅	虜	了	亮	僚	両	凌
4E4	寮	料	梁	涼	猟	療	瞭	稜	糧	良	諒	遼	量	陵	領	力	4E5	緑	倫	厘	林	淋	燐	琳	臨	輪	隣	鱗	麟	瑠	塁	涙	累
4E6	類	令	伶	例	冷	励	嶺	怜	玲	礼	苓	鈴	隷	零	霊	麗	4E7	齢	暦	歴	列	劣	烈	裂	廉	恋	憐	漣	煉	簾	練	聯	
4F2		蓮	連	錬	呂	魯	櫓	炉	賂	路	露	労	婁	廊	弄	朗	4F3	楼	榔	浪	漏	牢	狼	篭	老	聾	蝋	郎	六	麓	禄	肋	録
4F4	論	倭	和	話	歪	賄	脇	惑	枠	鷲	亙	亘	鰐	詫	藁	蕨	4F5	椀	湾	碗	腕												
4F6																	4F7																
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

索　引

パソコンPCシリーズ　９８０１・８８０１ハンドブック

昭和５８年７月１日発行　　定価１３００円
編者　　　朝日新聞社電子計算室
発行者　　小島　茂
印刷製本　凸版印刷株式会社
発行所　　朝日新聞社
〒１０４　東京都中央区築地５－３－２
電話　０３（５４５）０１３１（代表）
編集＝出版局プロジェクト室　　販売＝出版販売部
振替　東京０－１７３０

●本書の内容

©朝日新聞社 1983 2055-258144-0042 Printed in Japan

定価1300円

PC9801・8801ハンドブック
プロ入門のための機能徹底解説
朝日新聞電子計算室編
朝日新聞社